FUNDAMENTOS de BASES DE DATOS

SILBERSCHATZ • KORTH • SUDARSHAN

6.ª EDICIÓN

McGraw Hill Education

Fundamentos de bases de datos

Sexta edición

Abraham Silberschatz
Universidad de Yale

Henry F. Korth
Universidad de Lehigh

S. Sudarshan
Instituto Tecnológico Indio, Bombay

Revisión técnica
Jesús Sánchez Allende
Universidad Alfonso X El Sabio (Madrid)

McGraw Hill Education

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

Traducción: **Pilar Moreno Díaz**

Fundamentos de bases de datos, sexta edición

No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión por 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. Diríjase a CEDRO (Centro Español de Derechos Reprográficos, www.cedro.org) si necesita fotocopiar o escanear algún fragmento de esta obra.

Derechos reservados © 2014, respecto a la sexta edición en español, por:

McGraw-Hill/Interamericana de España, S.L.
Edificio Valrealty, 1.ª planta
Basauri, 17
28023 Aravaca (Madrid)

Traducido de la sexta edición en inglés de DATABASE SYSTEM CONCEPTS, by Silberschatz, Abraham, 6th ed. © 2011 by The McGraw-Hill Companies, Inc. Todos los derechos reservados.
ISBN: 978-0-07-352332-3

ISBN: 978-84-481-9033-0
Depósito legal: M-12103-2014

Editora: Cristina Sánchez Sáinz-Trápaga
Director Gerente Universidad y Profesional: Norberto Rosas Gómez
Director General España y Portugal: Álvaro García Tejeda
Diseño de cubierta: Cian Network
Composición: www.dfrente.es
Impresión: XXX

IMPRESO EN ESPAÑA- PRINTED IN SPAIN

En memoria de mi padre, Joseph Silberschatz, de mi madre, Vera Silberschatz, y de mis abuelos, Stepha y Aaron Rosenblum.

Avi Silberschatz

A mi esposa, Joan, mis hijos, Abigail y Joseph, y mis padres, Henry y Frances.

Hank Korth

A mi esposa, Sita, mis hijos, Madhur y Advaith, y mi madre, Indira.

S. Sudarshan

Contenidos

Capítulo 1. Introducción

1.1. Aplicaciones de los sistemas de bases de datos 1
1.2. Propósito de los sistemas de bases de datos 2
1.3. Visión de los datos 3
1.4. Lenguajes de bases de datos 5
1.5. Bases de datos relacionales 6
1.6. Diseño de bases de datos 7
1.7. Almacenamiento de datos y consultas 9
1.8. Gestión de transacciones 10
1.9. Arquitectura de las bases de datos 10
1.10. Minería y análisis de datos 12
1.11. Bases de datos específicas 12
1.12. Usuarios y administradores de bases de datos 13
1.13. Historia de los sistemas de bases de datos 13
1.14. Resumen 15
Términos de repaso 15
Ejercicios prácticos 16
Ejercicios 16
Herramientas 16
Notas bibliográficas 16

Parte 1. Bases de datos relacionales

Capítulo 2. Introducción al modelo relacional

2.1. La estructura de las bases de datos relacionales 19
2.2. Esquema de la base de datos 20
2.3. Claves 21
2.4. Diagramas de esquema 23
2.5. Lenguajes de consulta relacional 23
2.6. Operaciones relacionales 23
2.7. Resumen 25
Términos de repaso 25
Ejercicios prácticos 25
Ejercicios 26
Notas bibliográficas 26

Capítulo 3. Introducción a SQL

3.1. Introducción al lenguaje de consultas SQL 27
3.2. Definición de datos de SQL 27
3.3. Estructura básica de las consultas SQL 29
3.4. Operaciones básicas adicionales 34
3.5. Operaciones sobre conjuntos 36
3.6. Valores nulos 37
3.7. Funciones de agregación 38
3.8. Subconsultas anidadas 40
3.9. Modificación de la base de datos 43
3.10. Resumen 46
Términos de repaso 46
Ejercicios prácticos 46
Ejercicios 48
Herramientas 49
Notas bibliográficas 49

Capítulo 4. SQL intermedio

4.1. Expresiones de reunión 51
4.2. Vistas 55
4.3. Transacciones 57
4.4. Restricciones de integridad 58
4.5. Tipos de datos y esquemas de SQL 61
4.6. Autorización 64
4.7. Resumen 67
Términos de repaso 68
Ejercicios prácticos 68
Ejercicios 69
Notas bibliográficas 69

Capítulo 5. SQL avanzado

5.1. Acceso a SQL desde lenguajes de programación 71
5.2. Funciones y procedimientos 77
5.3. Disparadores 80
5.4. Consultas recursivas ** 83
5.5. Características de agregación avanzadas ** 85
5.6. OLAP** 88
5.7. Resumen 93
Términos de repaso 93
Ejercicios prácticos 93
Ejercicios 94
Notas bibliográficas 95
Herramientas 96

Capítulo 6. Lenguajes formales de consulta relacional

6.1. El álgebra relacional 97
6.2. El cálculo relacional de tuplas 107
6.3. El cálculo relacional de dominios 109
6.4. Resumen 110
Términos de repaso 111
Ejercicios prácticos 111
Ejercicios 112
Notas bibliográficas 113

Parte 2. Diseño de bases de datos

Capítulo 7. Diseño de bases de datos y el modelo E-R

7.1. Visión general del proceso de diseño 117
7.2. El modelo entidad-relación 118
7.3. Restricciones 121
7.4. Eliminar atributos redundantes de un conjunto de entidades 123
7.5. Diagramas entidad-relación 124
7.6. Reducción a esquemas relacionales 128
7.7. Aspectos del diseño entidad-relación 131
7.8. Características del modelo E-R extendido 133
7.9. Notaciones alternativas para el modelado de datos 137
7.10. Otros aspectos del diseño de bases de datos 140
7.11. Resumen 142
Términos de repaso 142
Ejercicios prácticos 143
Ejercicios 144
Herramientas 145
Notas bibliográficas 145

Capítulo 8. Diseño de bases de datos y el modelo E-R

8.1. Características de los buenos diseños relacionales 147
8.2. Dominios atómicos y la primera forma normal 149
8.3. Descomposición mediante dependencias funcionales 149
8.4. Teoría de las dependencias funcionales 153
8.5. Algoritmos de descomposición 157
8.6. Descomposición mediante dependencias multivaloradas 160
8.7. Más formas normales 161
8.8. Proceso de diseño de la base de datos 162
8.9. Modelado de datos temporales 163
8.10. Resumen 165
Términos de repaso 165
Ejercicios prácticos 166
Ejercicios 167
Notas bibliográficas 168

Capítulo 9. Diseño y desarrollo de aplicaciones

9.1. Interfaces de usuario y programas de aplicación 169
9.2. Fundamentos de la web 170
9.3. Servlets y JSP 173
9.4. Arquitecturas de aplicación 176
9.5. Desarrollo rápido de aplicaciones 179
9.6. Rendimiento de la aplicación 181
9.7. Seguridad de las aplicaciones 181
9.8. Cifrado y sus aplicaciones 185
9.9. Resumen 188
Términos de repaso 189
Ejercicios prácticos 189
Ejercicios 190
Sugerencias de proyectos 191
Herramientas 192
Notas bibliográficas 192

Parte 3. Almacenamiento de datos y consultas

Capítulo 10. Almacenamiento y estructura de archivos

10.1. Visión general de los medios físicos de almacenamiento 195
10.2. Discos magnéticos y almacenamiento flash 196
10.3. RAID 200
10.4. Almacenamiento terciario 205
10.5. Organización de los archivos 206
10.6. Organización de los registros en archivos 208
10.7. Almacenamiento con diccionarios de datos 210
10.8. Memoria intermedia de la base de datos 211
10.9. Resumen 213
Términos de repaso 213
Ejercicios prácticos 214
Ejercicios 215
Notas bibliográficas 215

Capítulo 11. Indexación y asociación

11.1. Conceptos básicos 217
11.2. Índices ordenados 217
11.3. Archivos de índices de árboles B^+ 222
11.4. Extensiones de los árboles B^+ 230
11.5. Accesos bajo varias claves 233
11.6. Asociación estática 234
11.7. Asociación dinámica 237
11.8. Comparación entre la indexación ordenada y la asociación 241
11.9. Índices de mapas de bits 242
11.10. Definición de índices en SQL 243
11.11. Resumen 244
Términos de repaso 245
Ejercicios prácticos 245
Ejercicios 246
Notas bibliográficas 247

Capítulo 12. Procesamiento de consultas

12.1. Descripción general 249
12.2. Medidas del coste de una consulta 250
12.3. Operación selección 251
12.4. Ordenación 253
12.5. Operación reunión 254
12.6. Otras operaciones 261
12.7. Evaluación de expresiones 262
12.8. Resumen 265
Términos de repaso 265
Ejercicios prácticos 266
Ejercicios 267
Notas bibliográficas 267

Capítulo 13. Optimización de consultas

13.1. Visión general 269
13.2. Transformación de expresiones relacionales 271
13.3. Estimación de las estadísticas de los resultados de las expresiones 274
13.4. Elección de los planes de evaluación 278
13.5. Vistas materializadas** 282
13.6. Temas avanzados de optimización de consultas** 284
13.7. Resumen 286
Términos de repaso 286
Ejercicios prácticos 287
Ejercicios 288
Notas bibliográficas 289

Parte 4. Gestión de transacciones

Capítulo 14. Transacciones

14.1. Concepto de transacción 293
14.2. Un modelo simple de transacciones 294
14.3. Estructura de almacenamiento 295
14.4. Atomicidad y durabilidad de las transacciones 295
14.5. Aislamiento de transacciones 297
14.6. Secuencialidad 299
14.7. Aislamiento y atomicidad de transacciones 301
14.8. Niveles de aislamiento de transacciones 302
14.9. Implementación de niveles de aislamiento 302
14.10. Transacciones como sentencias de SQL 304
14.11. Resumen 305
Términos de repaso 306
Ejercicios prácticos 306
Ejercicios 307
Notas bibliográficas 307

Capítulo 15. Control de concurrencia

15.1. Protocolos basados en el bloqueo 309
15.2. Tratamiento de interbloqueos 314
15.3. Granularidad múltiple 316
15.4. Protocolos basados en marcas temporales 318
15.5. Protocolos basados en validación 320
15.6. Esquemas multiversión 321
15.7. Aislamiento de instantáneas 322
15.8. Operaciones para insertar y borrar, y lectura de predicados 324
15.9. Niveles débiles de consistencia en la práctica 326
15.10. Concurrencia en las estructuras de índices** 328
15.11. Resumen 330
Términos de repaso 331
Ejercicios prácticos 332
Ejercicios 333
Notas bibliográficas 334

Capítulo 16. Sistema de recuperación

16.1. Clasificación de los fallos 335
16.2. Almacenamiento 335
16.3. Recuperación y atomicidad 337
16.4. Algoritmo de recuperación 341
16.5. Gestión de memoria intermedia 343
16.6. Fallo con pérdida de almacenamiento no volátil 345
16.7. Liberación rápida de bloqueos y operaciones de deshacer lógicas 345
16.8. ARIES** 349
16.9. Sistemas remotos de copias de seguridad 352
16.10. Resumen 353
Términos de repaso 354
Ejercicios prácticos 355
Ejercicios 356
Notas bibliográficas 357

Parte 5. Arquitectura de sistemas

Capítulo 17. Arquitecturas de los sistemas de bases de datos

17.1. Arquitecturas centralizadas y cliente-servidor 361
17.2. Arquitecturas de sistemas servidores 363
17.3. Sistemas paralelos 365
17.4. Sistemas distribuidos 368
17.5. Tipos de redes 370
17.6. Resumen 372
Términos de repaso 372
Ejercicios prácticos 373
Ejercicios 373
Notas bibliográficas 374

Capítulo 18. Bases de datos paralelas

18.1. Introducción 375
18.2. Paralelismo de E/S 375
18.3. Paralelismo entre consultas 377
18.4. Paralelismo en consultas 377
18.5. Paralelismo en operaciones 378
18.6. Paralelismo entre operaciones 382
18.7. Optimización de consultas 382
18.8. Diseño de sistemas paralelos 383
18.9. Paralelismo en procesadores multinúcleo 383
18.10. Resumen 384
Términos de repaso 385
Ejercicios prácticos 385
Ejercicios 386
Notas bibliográficas 387

Capítulo 19. Bases de datos distribuidas

19.1. Bases de datos homogéneas y heterogéneas 389
19.2. Almacenamiento distribuido de datos 389
19.3. Transacciones distribuidas 391
19.4. Protocolos de compromiso 392
19.5. Control de concurrencia en las bases de datos distribuidas 395
19.6. Disponibilidad 398
19.7. Procesamiento distribuido de consultas 402
19.8. Bases de datos distribuidas heterogéneas 403
19.9. Bases de datos en la nube 405
19.10. Sistemas de directorio 409
19.11. Resumen 412
Términos de repaso 413
Ejercicios prácticos 414
Ejercicios 415
Notas bibliográficas 415

Parte 6. Almacenes de datos, minería de datos y recuperación de la información

Capítulo 20. Almacenes de datos y minería de datos

20.1. Sistemas de ayuda a la toma de decisiones 419
20.2. Almacenes de datos 420
20.3. Minería de datos 422
20.4. Clasificación 422
20.5. Reglas de asociación 427
20.6. Otros tipos de asociación 428
20.7. Agrupamiento 428
20.8. Otros tipos de minería 429
20.9. Resumen 429
Términos de repaso 430
Ejercicios prácticos 430
Ejercicios 431
Herramientas 431
Notas bibliográficas 431

Capítulo 21. Recuperación de información

21.1. Descripción general 433
21.2. Clasificación por relevancia según los términos 434
21.3. Relevancia según los hipervínculos 435
21.4. Sinónimos, homónimos y ontologías 438
21.5. Creación de índices de documentos 439
21.6. Medida de la efectividad de la recuperación 439
21.7. Robots de búsqueda e indexación en web 440
21.8. Recuperación de información: más allá de la clasificación de las páginas 441
21.9. Directorios y categorías 442
21.10. Resumen 444
Términos de repaso 444
Ejercicios prácticos 445
Ejercicios 445
Herramientas 445
Notas bibliográficas 446

Parte 7. Bases de datos especiales

Capítulo 22. Bases de datos orientadas a objetos

22.1. Descripción general 449
22.2. Tipos de datos complejos 449
22.3. Tipos estructurados y herencia en SQL 451
22.4. Herencia de tablas 452
22.5. Tipos array y multiconjunto en SQL 453
22.6. Identidad de los objetos y tipos de referencia en SQL 455
22.7. Implementación de las características O-R 456
22.8. Lenguajes de programación persistentes 457
22.9. Correspondencia entre el modelo relacional y el orientado a objetos 461
22.10. Sistemas orientados a objetos y sistemas relacionales orientados a objetos 461
22.11. Resumen 462
Términos de repaso 462
Ejercicios prácticos 463
Ejercicios 463
Herramientas 464
Notas bibliográficas 464

Capítulo 23. XML

23.1. Motivación 465
23.2. Estructura de los datos XML 467
23.3. Esquema de documentos XML 469
23.4. Consulta y transformación 472
23.5. La interfaz de programación de aplicaciones de XML 476
23.6. Almacenamiento de datos XML 476
23.7. Aplicaciones XML 479
23.8. Resumen 481
Términos de repaso 481
Ejercicios prácticos 482
Ejercicios 483
Herramientas 483
Notas bibliográficas 484

Parte 8. Temas avanzados

Capítulo 24. Desarrollo avanzado de aplicaciones

24.1. Ajuste del rendimiento 487
24.2. Pruebas de rendimiento 494
24.3. Otros temas sobre el desarrollo de aplicaciones 496
24.4. Normalización 497
24.5. Resumen 500
Términos de repaso 500
Ejercicios prácticos 501
Ejercicios 501
Notas bibliográficas 502

Capítulo 25. Datos espaciales, temporales y movilidad

25.1. Motivación 503
25.2. El tiempo en las bases de datos 503
25.3. Datos espaciales y geográficos 504
25.4. Bases de datos multimedia 510
25.5. Movilidad y bases de datos personales 511
25.6. Resumen 514
Términos de repaso 515
Ejercicios prácticos 515
Ejercicios 516
Notas bibliográficas 516

Capítulo 26. Procesamiento avanzado de transacciones

26.1. Monitores de procesamiento de transacciones 517
26.2. Flujos de trabajo de transacciones 519
26.3. Comercio electrónico 522
26.4. Bases de datos en memoria principal 524
26.5. Sistemas de transacciones de tiempo real 525
26.6. Transacciones de larga duración 525
26.7. Resumen 528
Términos de repaso 529
Ejercicios prácticos 530
Ejercicios 530
Notas bibliográficas 531

Parte 9. Estudio de casos

Capítulo 27. PostgreSQL

27.1. Introducción 535
27.2. Interfaces de usuario 535
27.3. Variaciones y extensiones de SQL 537
27.4. Gestión de transacciones en PostgreSQL 541
27.5. Almacenamiento e índices 545
27.6. Procesamiento y optimización de consultas 547
27.7. Arquitectura del sistema 549
Notas bibliográficas 550

Capítulo 28. Oracle

28.1. Herramientas para el diseño y la consulta de bases de datos 551
28.2. Variaciones y extensiones de SQL 552
28.3. Almacenamiento e indexación 553
28.4. Procesamiento y optimización de consultas 558
28.5. Control de concurrencia y recuperación 561
28.6. Arquitectura del sistema 563
28.7. Réplica, distribución y datos externos 565
28.8. Herramientas de gestión de bases de datos 566
28.9. Minería de datos 567
Notas bibliográficas 567

Capítulo 29. DB2 Universal Database de IBM

29.1. Visión general 569
29.2. Herramientas de diseño de bases de datos 570
29.3. Variaciones y extensiones de SQL 570
29.4. Almacenamiento e indexación 572
29.5. Agrupación multidimensional 574
29.6. Procesamiento y optimización de consultas 575
29.7. Tablas de consultas materializadas 577
29.8. Características autónomas de B2 578
29.9. Herramientas y utilidades 579
29.10. Control de concurrencia y recuperación 579
29.11. Arquitectura del sistema 581
29.12. Réplicas, distribución y datos externos 581
29.13. Características de inteligencia de negocio 581
Notas bibliográficas 583

Capítulo 30. Microsoft SQL Server

30.1. Herramientas para la administración, el diseño y la consulta de las bases de datos 585
30.2. Variaciones y extensiones de SQL 587
30.3. Almacenamiento e índices 589
30.4. Procesamiento y optimización de consultas 591
30.5. Concurrencia y recuperación 593
30.6. Arquitectura del sistema 595
30.7. Acceso a los datos 597
30.8. Procesamiento de consultas heterogéneas distribuidas 597
30.9. Réplica 598
30.10. Programación de servidores en .NET 599
30.11. Soporte de XML 601
30.12. Service Broker de SQLServer 603
30.13. Inteligencia de negocio 604
Notas bibliográficas 606

Parte 10. Apéndices

• Apéndice A 609
• Bibliografía 615
• Índice analítico 629

Prefacio

La gestión de las bases de datos ha evolucionado desde una aplicación informática especializada hasta convertirse en parte esencial de los entornos informáticos modernos. Por tanto, el conocimiento acerca de los sistemas de bases de datos se ha convertido en una parte imprescindible de la formación en Informática. En este texto se presentan los conceptos fundamentales de la gestión de las bases de datos. Estos conceptos incluyen aspectos del diseño, de los lenguajes y de la implementación de los sistemas de bases de datos.

Este libro está orientado a un primer curso de bases de datos para los niveles técnicos y superiores. Además del material básico para un primer curso, el texto también contiene material avanzado que se puede usar como complemento del curso o como material introductorio de un curso avanzado.

En este libro se asume que se dispone de conocimientos básicos sobre estructuras de datos básicas, organización de computadoras y un lenguaje de programación como Java, C o Pascal. Los conceptos se presentan usando descripciones intuitivas, muchas de las cuales están basadas en el ejemplo propuesto de una universidad. Se tratan los resultados teóricos importantes, pero se omiten las demostraciones formales. En lugar de las demostraciones se usan figuras y ejemplos para sugerir su justificación. Las descripciones formales y las pruebas pueden hallarse en los artículos de investigación y en los textos avanzados a los que se hace referencia en las notas bibliográficas.

Los conceptos y algoritmos fundamentales tratados en este libro suelen basarse en los utilizados en sistemas de bases de datos comerciales o experimentales ya existentes. El objetivo es presentar esos conceptos y algoritmos en un marco general que no esté vinculado a ningún sistema de bases de datos en concreto. Los detalles de los sistemas de bases de datos concretos se estudian en la Parte 9, «Estudio de casos».

En esta sexta edición de *Fundamentos de bases de datos* se ha mantenido el estilo global de las ediciones anteriores, mientras que su contenido y organización han evolucionado para reflejar las modificaciones que se están produciendo en el modo en que las bases de datos se diseñan, se gestionan y se usan. También se han tenido en cuenta las tendencias en la enseñanza de los conceptos de bases de datos y se han hecho adaptaciones para facilitar esas tendencias donde ha resultado conveniente.

Organización

El texto está organizado en nueve partes principales y cinco apéndices que conforman la última parte (solo el Apéndice A está incluido en el libro; los Apéndices B, C, D y E están en red en inglés: http://www.db.book.com).

- **Visión general** (Capítulo 1). En el Capítulo 1 se proporciona una visión general de la naturaleza y propósito de los sistemas de bases de datos. Se explica cómo se ha desarrollado el concepto de sistema de bases de datos, cuáles son las características usuales de los sistemas de bases de datos, lo que proporciona al usuario un sistema de bases de datos y cómo se comunican los sistemas de bases de datos con los sistemas operativos. También se introduce un ejemplo de aplicación de las bases de datos: la organización de una universidad que consta de múltiples departamentos, profesores, estudiantes y cursos. Este ejemplo se usa a lo largo de todo el libro. Este capítulo es de naturaleza justificativa, histórica y explicativa.
- **Parte 1: Bases de datos relacionales** (Capítulos 2 a 6). El Capítulo 2 introduce el modelo de datos relacional y trata de conceptos básicos, como la estructura del álgebra relacional, claves, diagramas de esquema, lenguajes de consulta relacional y operaciones relacionales. Los Capítulos 3, 4 y 5 se centran en el más influyente de los lenguajes relacionales orientados al usuario. El Capítulo 6 trata los lenguajes de consulta relacional formales: el álgebra relacional, el cálculo relacional de tuplas y el cálculo relacional de dominios.

  Los capítulos de esta parte del libro describen la manipulación de los datos: consultas, actualizaciones, inserciones y eliminaciones, y dan por supuesto que se ha proporcionado un diseño de esquema. Los aspectos del diseño de esquemas se posponen hasta la Parte 2.
- **Parte 2: Diseño de bases de datos** (Capítulos 7 a 9). El Capítulo 7 ofrece una visión general del proceso de diseño de las bases de datos, con el énfasis puesto en el diseño mediante el modelo de datos entidad-relación. Este modelo ofrece una vista de alto nivel de los aspectos del diseño de las bases de datos y de los problemas que se encuentran al capturar la semántica de las aplicaciones realistas en las restricciones de un modelo de datos. La notación de los diagramas de clase UML también se trata en este capítulo.

  El Capítulo 8 introduce la teoría del diseño de las bases de datos relacionales. Se tratan la teoría de las dependencias funcionales y de la normalización, con el énfasis puesto en la motivación y la comprensión intuitiva de cada forma normal. Este capítulo comienza con una visión general del diseño relacional y se basa en la comprensión intuitiva de la implicación lógica de las dependencias funcionales. Esto permite introducir el concepto de normalización antes de haber tratado completamente la teoría de la dependencia funcional, que se presenta más avanzado el capítulo. Los profesores pueden decidir usar únicamente este tratamiento inicial de los Secciones 8.1 a 8.3 sin pérdida de continuidad. Los profesores que empleen todo el capítulo conseguirán que los estudiantes tengan una buena comprensión de los conceptos de normalización para justificar algunos de los conceptos más difíciles de comprender de la teoría de la dependencia funcional.

El Capítulo 9 trata del diseño y del desarrollo de las aplicaciones. Este capítulo pone énfasis en la creación de aplicaciones de bases de datos con interfaces basadas en web. Además, el capítulo trata de la seguridad de las aplicaciones.

- **Parte 3: Almacenamiento de datos y consultas** (Capítulos 10 a 13). El Capítulo 10 trata de dispositivos de almacenamiento, archivos y estructuras de almacenamiento de datos. En el Capítulo 11 se presenta una gran variedad de técnicas de acceso a los datos, incluidos los índices asociativos y de árbol B^+. Los Capítulos 12 y 13 abordan los algoritmos de evaluación de consultas y su optimización. En estos capítulos se examinan los aspectos internos de los componentes de almacenamiento y de recuperación de las bases de datos.
- **Parte 4: Gestión de transacciones** (Capítulos 14 a 16). El Capítulo 14 se centra en los fundamentos de los sistemas de procesamiento de transacciones: la atomicidad, la consistencia, el aislamiento y la durabilidad. Proporciona una descripción general de los métodos que se utilizan para garantizar estas propiedades, como el bloqueo y la creación de vistas.

  El Capítulo 15 se centra en el control de la concurrencia y presenta varias técnicas para garantizar la secuencialidad, incluidos el bloqueo, las marcas de tiempo y las técnicas de optimización (de validación). Este capítulo también trata los problemas asociados a los bloqueos. También se trata con detalle la serialización, utilizada generalmente en la creación de vistas.

  El Capitulo 16 trata las principales técnicas para garantizar la ejecución correcta de las transacciones pese a las caídas del sistema y los fallos de los discos. Estas técnicas incluyen los registros, los puntos de revisión y los volcados de las bases de datos. Se presenta el ampliamente utilizado algoritmo ARIES.
- **Parte 5: Arquitectura de sistemas** (Capítulos 17 a 19). El Capítulo 17 trata de la arquitectura de los sistemas informáticos y describe la influencia de los subyacentes a los sistemas de bases de datos. En este capítulo se estudian los sistemas centralizados, los sistemas cliente - servidor y las arquitecturas paralela y distribuida.

  El Capítulo 18, que trata las bases de datos paralelas, explora una gran variedad de técnicas de paralelismo, incluidos el paralelismo de E/S, el paralelismo en consultas y entre consultas, y el paralelismo en operaciones y entre operaciones. Este capítulo también describe el diseño de sistemas paralelos.

  El Capítulo 19 trata de los sistemas distribuidos de bases de datos, revisitando los aspectos del diseño de bases de datos, la gestión de las transacciones y la evaluación y la optimización de las consultas en el contexto de las bases de datos distribuidas. Este capítulo también trata aspectos de la disponibilidad de los sistemas durante los fallos, las bases de datos distribuidas heterogéneas, las bases de datos basadas en la nube y los sistemas de directorio distribuidos.
- **Parte 6: Almacenes de datos, minería de datos y recuperación de la información** (Capítulos 20 y 21). El Capítulo 20 introduce el concepto de almacén de datos y minería de datos. El Capítulo 21 describe las técnicas de recuperación de información para consultas de datos textuales, incluyendo las técnicas basadas en hiperenlaces de los motores de búsqueda en web.

  La Parte 6 usa los conceptos de modelado y de lenguaje de las Partes 1 y 2, pero no depende de las Partes 3, 4 o 5. Por tanto, puede incorporarse fácilmente en cursos que se centren en SQL y en el diseño de bases de datos.
- **Parte 7: Bases de datos especiales** (Capítulos 22 a 23). El Capítulo 22 trata las bases de datos orientadas a objetos. El capítulo describe el modelo de datos relacional orientado a objetos, que extiende el modelo relacional para admitir tipos de datos complejos, la herencia de tipos y la identidad de objetos. El capítulo también describe el acceso a las bases de datos mediante lenguajes de programación orientados a objetos.

  El Capítulo 23 trata el estándar XML para la representación de datos, que cada vez se utiliza más en el intercambio y almacenamiento de datos complejos. El capítulo también describe los lenguajes de consulta para XML.
- **Parte 8: Temas avanzados** (Capítulos 24 a 26). El Capítulo 24 trata temas avanzados del desarrollo de aplicaciones, incluyendo el análisis de rendimiento, las pruebas de rendimiento, la prueba de aplicaciones de bases de datos y la estandarización.

  El Capítulo 25 trata sobre datos geográficos y espaciales, datos temporales, datos multimedia y sobre las bases de datos personales y móviles.

  Finalmente, el Capítulo 26 trata del procesamiento avanzado de transacciones. Entre los temas tratados están los monitores de procesamiento de transacciones, los flujos de trabajo transaccionales, el comercio electrónico, los sistemas de transacciones de alto rendimiento, los sistemas de transacciones en tiempo real y las transacciones de larga duración.
- **Parte 9: Estudio de casos** (Capítulos 27 a 30). En esta parte se estudian cuatro de los principales sistemas de bases de datos, como PostgreSQL, Oracle, DB2 de IBM y SQL Server de Microsoft. Estos capítulos destacan las características propias de cada uno de los sistemas y describen su estructura interna. Ofrecen gran abundancia de información interesante sobre los productos respectivos y ayudan al lector a comprender el uso en los sistemas reales de las diferentes técnicas de implementación descritas en partes anteriores. También tratan varios aspectos prácticos interesantes del diseño de sistemas reales.
- **Apéndices.** El libro incluye cinco apéndices que tratan material de naturaleza histórica o temas más avanzados; estos apéndices solo están disponibles en línea en el sitio web del libro (http://www.db-book.com). La excepción es el Apéndice A, que presenta con detalle el esquema del ejemplo de la universidad incluyendo el esquema completo, el LDD y todas las tablas. Este apéndice se encuentra al final del libro.

  El Apéndice B describe otros lenguajes de consulta relacionales, incluyendo QBE Microsoft Access y Datalog.

  El Apéndice C describe el diseño de bases de datos relacionales, incluyendo la teoría de dependencias multivaloradas, las dependencias de reunión, las proyecciones y las formas normales dominio-clave. Este apéndice es una ayuda para los que quieren estudiar la teoría del diseño de bases de datos relacionales con más detalle y para los profesores que quieren incluirla en sus cursos. Este apéndice está disponible exclusivamente en línea, en el sitio web del libro.

  Aunque la mayoría de las aplicaciones de bases de datos utilizan el modelo relacional o el modelo relacional orientado a objetos, también se utilizan los modelos de datos jerárquicos y en red en algunas aplicaciones antiguas. Para los lectores que quieran conocer más detalles de estos modelos de datos se proporcionan apéndices que describen estos modelos de datos en los Apéndices D y E, respectivamente.

La sexta edición

La producción de esta sexta edición ha estado guiada por los muchos comentarios y sugerencias que se han recibido relativos a ediciones anteriores, por nuestras propias observaciones al ejercer la docencia en la Universidad de Yale, la Universidad de Lehigh, el IIT de Bombay y por nuestro análisis de las direcciones hacia las que está evolucionando la tecnología de las bases de datos.

Se ha sustituido el ejemplo anterior de un banco por el de una universidad. Este ejemplo tiene una relación intuitiva inmediata

con los estudiantes que les ayudará a recordar el ejemplo y, aún más importante, a obtener una mejor comprensión de las decisiones de diseño que se han de realizar.

Se ha reorganizado el libro para poner agrupado todo lo que se trataba sobre SQL en la primera parte del libro. Los Capítulos 3, 4 y 5 presentan completamente SQL. En el Capítulo 3 se tratan los principios del lenguaje y en el Capítulo 4 los aspectos más avanzados. En el Capítulo 5 se presenta JDBC junto con otros mecanismos de acceso SQL desde lenguajes de programación de propósito general. Se presentan los disparadores y la recursión y se termina con el procesamiento analítico en línea (OLAP). Los cursos de introducción pueden preferir tratar solamente algunas de las secciones del Capítulo 5 o saltarse secciones hasta después de tratar el diseño de bases de datos sin perder continuidad.

Además de los cambios anteriores se ha revisado el material de todos los capítulos, actualizándolo, añadiendo temas más recientes de la tecnología de bases de datos y mejorando las descripciones de temas que los estudiantes suelen tener problemas para entender. También se han añadido nuevos ejercicios y se han actualizado las referencias. La lista de cambios incluye los siguientes:

- **SQL anterior.** Muchos profesores utilizan SQL como un componente clave para sus prácticas (consulte el sitio web, http://www.db-book.com, para ver proyectos de ejemplo). Para que los estudiantes tengan suficiente tiempo para sus proyectos, en concreto para universidades, es esencial enseñar SQL lo antes posible. Con esta idea en mente, se han realizado varios cambios en la organización:
  - Un nuevo capítulo sobre el modelo relacional (Capítulo 2) que precede a SQL, con los fundamentos conceptuales sin perderse en los detalles del álgebra relacional.
  - Los Capítulos 3, 4 y 5 tratan SQL en detalle. Estos capítulos también tratan otras variantes de distintos sistemas de bases de datos, para minimizar los problemas que deben afrontar los estudiantes cuando ejecutan consultas en sistemas de bases de datos reales. Estos capítulos tratan todos los aspectos de SQL, incluyendo consultas, definición de datos, especificación de restricciones, OLAP y el uso de SQL desde distintos lenguajes, entre ellos Java/JDBC.
  - Los lenguajes formales se han pospuesto (Capítulo 6) tras SQL, y se pueden omitir sin afectar a la secuencialidad del resto de los capítulos. Solamente el tema de la optimización de consultas del Capítulo 13 depende del tema de álgebra relacional del Capítulo 6.
- **Nuevo esquema de base de datos.** Se ha adoptado un nuevo esquema, que utiliza los datos de una universidad, como ejemplo para todo el libro. Este esquema es más intuitivo y motivador para los estudiantes que el anterior de un banco y muestra compromisos de diseño más complejos en los capítulos de diseño de bases de datos.
- **Más material con el que experimentar los estudiantes.** Para facilitar que se pueda seguir el ejemplo del libro se presenta el esquema de la base de datos y ejemplos de relaciones en el Apéndice A así como donde se usa en otros capítulos. Además, en el sitio web http://www.db-book.com se dispone de las sentencias SQL de definición de datos de todo el ejemplo, así como sentencias SQL para crear todas las relaciones de ejemplo. De esta forma los estudiantes pueden ejecutar sus consultas directamente en un sistema de bases de datos y modificarlas.
- **Material del modelo E-R revisado.** La notación de diagramas E-R del Capítulo 7 se ha modificado para hacerlo compatible con UML. El capítulo también hace uso del esquema de base de datos de la universidad para mostrar compromisos de diseño más complejos.
- **Revisión del diseño relacional.** El Capítulo 8 se ha reescrito para que sea más legible, mejorando la forma de abordar las dependencias funcionales y la normalización antes de tratar la teoría de dependencias funcionales; de esta forma resulta más motivadora la teoría.
- **Más material sobre el desarrollo de aplicaciones y la seguridad.** El Capítulo 9 ofrece nuevo material sobre el desarrollo de aplicaciones, mostrando los cambios que se están produciendo en este campo. En particular se ha expandido en el tema de la seguridad considerando su criticidad en el actual mundo interconectado, con un énfasis especial en los temas prácticos frente a los abstractos.
- **Material revisado y actualizado sobre el almacenamiento de datos, el indexado y la optimización de consultas.** El Capítulo 10 se ha actualizado para incluir nuevas tecnologías, incluyendo las memorias flash. El tratamiento sobre los árboles B^+ se ha revisado en el Capítulo 11 para mostrar implementaciones prácticas, como la carga en bruto, y se ha mejorado la presentación. Se ha revisado el ejemplo sobre árboles B^+ con $n=4$, para evitar los casos especiales de nodos vacíos que se producían con el valor (no realista) de $n=3$.

  El Capítulo 13 tiene nuevo material sobre técnicas avanzadas de optimización de consultas.
- **Material revisado sobre gestión de transacciones.** El Capítulo 14 proporciona lo básico para un curso introductorio, dejando los aspectos más avanzados para los Capítulos 15 y 16. El Capítulo 14 se ha extendido para incluir temas prácticos de gestión de transacciones a los que se enfrentan los usuarios y los desarrolladores de bases de datos. Este capítulo también incluye una descripción general de los temas que se tratan en los Capítulos 15 y 16, asegurando que si se omiten los Capítulos 15 y 16 los estudiantes obtienen los conocimientos básicos sobre los conceptos de control de concurrencia y recuperación.

  Los Capítulos 14 y 15 incluyen una detallada exposición sobre el aislamiento de instantáneas, que se utilizan ampliamente en la actualidad, incluyendo sus potenciales problemas de uso.

  El Capítulo 16 incluye ahora una descripción simplificada de la recuperación basada en registros (log), tratando el algoritmo ARIES.
- **Material revisado y extendido sobre bases de datos distribuidas.** Se incluye el almacenamiento en la nube, que está teniendo cada vez un mayor interés para las aplicaciones empresariales. El almacenamiento en la nube ofrece a las empresas la oportunidad de mejorar sus costes de gestión y la escalabilidad del almacenamiento, en particular para las aplicaciones web. Se tratan estas ventajas junto a los riesgos y potenciales inconvenientes.

  Las bases de datos múltiples, que se trataban anteriormente en el capítulo sobre procesamiento de transacciones, ahora se tratan como parte del capítulo sobre bases de datos distribuidas.
- **Se ha pospuesto el material de bases de datos orientadas a objetos y XML.** Aunque los lenguajes orientados a objetos y XML se usan ampliamente junto con las bases de datos, su uso en las bases de datos aún es limitado, por lo que resulta apropiado para cursos avanzados o como material suplementario. Estos temas se han pospuesto en el libro a los Capítulos 22 y 23.
- **QBE, Microsoft Access y Datalog en un apéndice en línea.** Estos temas, que anteriormente se encontraban en el capítulo sobre «otros lenguajes relacionales», se han trasladado al Apéndice C, en línea.

Todos los temas no indicados anteriormente se han actualizado, aunque su organización se ha mantenido básicamente sin cambios.

Material y ejercicios

Cada capítulo tiene una lista de términos de repaso, además de un resumen, que puede ayudar al lector a revisar los temas clave de cada capítulo.

Los ejercicios se dividen en dos conjuntos: **ejercicios prácticos** y **ejercicios.** Puede encontrar las soluciones a los ejercicios prácticos en el sitio web del libro. Se alienta a los estudiantes a que resuelvan los ejercicios prácticos por su cuenta y, después, utilicen las soluciones del sitio web para comprobar sus propias soluciones. Las soluciones al resto de ejercicios solo están disponibles para los profesores (consulte «Notas para profesores», para más información sobre cómo obtener las soluciones).

Muchos de los capítulos tienen una sección de herramientas al final del capítulo que proporcionan información sobre herramientas de software relacionadas con el tema del capítulo; algunas de estas herramientas se pueden utilizar para realizar ejercicios de laboratorio. El LDD de SQL y los datos de ejemplo de la base de datos de la universidad y otras relaciones que se utilizan en los ejercicios están disponibles en el sitio web del libro, y se pueden utilizar para los ejercicios de laboratorio.

Nota para los profesores

Este libro contiene material tanto básico como avanzado, que puede que no se abarque en un solo semestre. Se han marcado varios apartados como avanzados mediante el símbolo «∗∗». Estos apartados pueden omitirse, si se desea, sin pérdida de continuidad. Los ejercicios que son difíciles (y pueden omitirse) también están marcados mediante el símbolo «∗∗».

Es posible diseñar los cursos usando varios subconjuntos de los capítulos. Ciertos capítulos también se pueden tratar en un orden diferente al orden del libro. Algunas de estas posibilidades son las siguientes:

- El Capítulo 5 (SQL avanzado) se puede saltar o retrasar sin pérdida de continuidad. Es de esperar que cursos más avanzados traten al menos la Sección 5.1.1, ya que JDBC es probable que sea una buena herramienta para los proyectos de los estudiantes.
- El Capítulo 6 (Lenguajes formales relacionales de consulta) se puede tratar inmediatamente tras el Capítulo 2, después de SQL. Alternativamente, este capítulo se puede omitir en un curso de introducción.

  Se recomienda tratar la Sección 6.1 (algebra relacional) si el curso también incluye el procesamiento de consultas. Sin embargo, se pueden omitir las Secciones 6.2 y 6.3 si no se va a utilizar el cálculo relacional en el curso.
- El Capítulo 7 (Modelo E-R) puede verse después de los Capítulos 3, 4 y 5 si así se desea, ya que el Capítulo 7 no tiene ninguna dependencia de SQL.
- El Capítulo 13 (Optimización de consultas) se puede omitir de un curso de fundamentos sin afectar a ningún otro capítulo.
- Tanto el procesamiento de transacciones (Capítulos 14 a 16) como la arquitectura de sistemas (Capítulos 17 a 19) constan de un capítulo de descripción general (Capítulos 14 y 17, respectivamente), seguidos por capítulos de detalle. Se puede elegir utilizar los Capítulos 14 y 17 y omitir los Capítulos 15, 16, 18 y 19, trasladándolos a un curso avanzado.
- Los Capítulos 20 y 21 tratan sobre los almacenes de datos, la minería de datos y la recuperación de la información; se pueden utilizar como material de estudio personal y omitirlos en un curso de fundamentos.
- Los Capítulos 22 (Bases de datos orientadas a objetos) y 23 (XML) se pueden omitir en un curso de fundamentos.
- Los Capítulos 24 a 26, que tratan sobre el desarrollo avanzado de aplicaciones, datos espaciales, temporales y móviles y el procesamiento avanzado de transacciones, son más apropiados para un curso avanzado o como material de estudio personal.
- Los casos de estudio de los Capítulos 27 a 30 son más apropiados para el estudio personal. Como alternativa, se pueden utilizar para ilustrar los conceptos de capítulos anteriores presentados en clase.

En el sitio web del libro puede encontrar modelos de ejemplo de programas de cursos: http://www.db-book.com.

Para el personal docente también se dispone del siguiente material adicional en: http://www.mhhe.com/silberschatz.

Contacto

Hemos puesto nuestros mayores esfuerzos en evitar errores tipográficos, lógicos y de otro tipo en el texto. Pero, como en cualquier nueva versión de software, seguramente existan algunos errores; en el sitio web del libro se encuentra accesible una lista de errores actualizada. Agradeceríamos que nos notificase cualquier error u omisión detectada en el libro que no se encuentre en la lista de erratas.

Estaríamos muy agradecidos de recibir sugerencias o mejoras al libro. También agradecemos cualquier contribución al sitio web del libro que pueda servir a otros lectores, como ejercicios de programación, sugerencias de proyectos, laboratorios y tutoriales en línea o trucos de enseñanza.

El correo electrónico de contacto es db-book-authors@cs.yale.edu. Cualquier otra correspondencia se debería enviar a Avi Silberschatz, Department of Computer Science, Yale University, 51 Prospect Street, P.O. Box 208285, New Haven, CT 06520-8285 USA.

Agradecimientos

Son muchas las personas que nos han ayudado en esta sexta edición, al igual que en las cinco ediciones anteriores.

Sexta edición

- Anastassia Ailamaki, Sailesh Krishnamurthy, Spiros Papadimitriou y Bianca Schroeder (Carnegie Mellon University) por escribir el Capítulo 27 describiendo el sistema de bases de datos PostgreSQL.
- Hakan Jakobsson (Oracle) por escribir el Capítulo 28 sobre el sistema de bases de datos Oracle.
- Sriram Padmanabhan (IBM) por escribir el Capítulo 29 sobre el sistema de bases de datos IBM DB2.
- Sameet Agarwal, José A. Blakeley, Thierry D'Hers, Gerald Hinson, Dirk Myers, Vaqar Pirzada, Bili Ramos, Balaji Rathakrishnan, Michael Rys, Florian Waas y Michael Zwilling (Microsoft) por escribir el Capítulo 30 describiendo el sistema de bases de datos Microsoft SQL Server, y en particular a José Blakeley por coordinar la edición del capítulo; César Galindo-Legaria, Goetz Graefe, Kalen Delaney y Thomas Casey (Microsoft) por sus contribuciones a la edición anterior del capítulo sobre Microsoft SQL.
- Daniel Abadi por la revisión de la tabla de contenidos de la quinta edición y su ayuda en la nueva organización.
- Steve Dolins, Universidad de Florida; Rolando Fernández, Universidad George Washington; Frantisek Franek, Universidad McMaster; Latifur Khan, Universidad de Texas-Dallas; Sanjay Madria, Universidad de Missouri-Roila; Aris Ouksel, Universidad de Illinois, y Richard Snodgrass, Universidad de Waterloo; revisores del libro y cuyos comentarios nos ayudaron en esta sexta edición.

- Judi Paige por su ayuda en la generación de las figuras y su presentación.
- Mark Wogahn por el uso del software para generar el libro, incluyendo las macros y fuentes de LaTeX, para que funcionasen correctamente.
- N. L. Sarda por sus comentarios, que nos ayudaron a mejorar varios capítulos, en particular el Capítulo 11; Vikram Pudi por motivarnos a sustituir el ejemplo anterior del banco, y Shetal Shah por sus comentarios en diversos capítulos.
- A los estudiantes de Yale, Lehigh y IIT Bombay por sus comentarios sobre la quinta edición, así como sobre las pruebas de esta sexta edición.

Ediciones anteriores

- Chen Li y Sharad Mehrotra por proporcionar el material sobre JDBC y seguridad para la quinta edición.
- Marilyn Turnamian y Nandprasad Joshi proporcionaron la ayuda de secretaría en la quinta edición y Marilyn también preparó un primer borrador del diseño de la cubierta de la quinta edición.
- Lyn Dupré realizó la edición de la tercera edición y Sara Strandtman editó el texto de la tercera edición.
- Nilesh Dalvi, Sumit Sanghai, Gaurav Bhalotia, Arvind Hulgeri K. V. Raghavan, Prateek Kapadia, Sara Strandtman, Greg Speegle y Dawn Bezviner ayudaron a preparar el manual de enseñanza de ediciones previas.
- La idea de utilizar barcos en la cubierta fue una idea que originalmente sugirió Bruce Stephan.
- Las siguientes personas indicaron errores de la quinta edición: Alex Coman, Ravindra Guravannavar, Arvind Hulgeri, Rohit Kulshreshtha, Sang-Won Lee, Joe H. C. Lu, Ahex N. Napitupulu, H. K. Park, Jian Pei, Fernando Sáenz Pérez, Donnie Pinkston, Yma Pinto, Rajarshi Rakshit, Sandeep Satpal, Amon Seagull, Barry Soroka, Praveen Ranjan Sriyastava, Hans Svensson, Moritz Wiese y Eyob Delehe Yirdaw.
- Las siguientes personas aportaron sugerencias y comentarios a la quinta y anteriores ediciones del libro. R. B. Abhyankar, Hani Abu-Salem, Jamel R. Alsabbagh, Raj Ashar, Don Batory, Phil Bernhard, Christian Breimann, Gavin M. Bierman, Janek Bogucki, Haran Boral, Paul Bourgeois, Phil Bohannon, Robert Brazile, Yuri Breitbart, Ramzi Bualuan, Michael Carey, Soumen Chakrabarti, Tom Chappell, Zhengxin Chen, Y. C. Chin, Jan Chomicki, Laurens Damen, Prasanna Dhandapani, Qin Ding, Valentin Dinu, J. Edwards, Christos Faloutsos, Homma Farian, Alan Fekete, Frantisek Franek, Shashi Gadia, Héctor García-Mohina, Goetz Graefe, Jim Gray, Le Gruenwald, Eitan M. Curan, William Hankley, Bruce Hillyer, Ron Hitchens, Chad Hogg, Arvind Hulgeri, Yannis Ioannidis, Zheng Jiaping, Randy M. Kaplan, Graham J. L. Kemp, Rami Khouri, Hyoung-Joo Kim, Won Kim, Henry Korth (padre de Henry F.), Carol Kroll, Hae Choon Lee, Sang-Won Lee, Irwin Levinstein, Mark Llewellyn, Gary Lindstrom, Ling Liu, Dave Maier, Keith Marzullo, Marty Maskarinec, Fletcher Mattox, Sharad Mehrotra, Jim Melton, Alberto Mendelzon, Ami Motro, Bhagirath Narahari, Yiu-Kai Dermis Ng, Thanh-Duy Nguyen, Anil Nigam, Cyril Orji, Meral Ozsoyoglu, D. B. Phatak, Juan Altmayer Pizzorno, Bruce Porter, Sunil Prabhakar, Jim Peterson, K. V. Raghavan, Nahid Rahman, Rajarshi Rakshit, Krithi Ramamritham, Mike Reiter, Greg Riccardi, Odinaldo Rodriguez, Mark Roth, Marek Rusinkiewicz, Michael Rys, Sunita Sarawagi, N. L. Sarda, Patrick Schmid, Nikhil Sethi, S. Seshadri, Stewart Shen, Shashi Shekhar, Amit Sheth, Max Smolens, Nandit Soparkar, Greg Speegle, Jeff Storey, Dilys Thomas, Prem Thomas, Tim Wahls, Anita Whitehall, Christopher Wilson, Marianne Winslett, Weining Zhang y Liu Zhenming.

Producción del libro

El editor fue Raghu Srinivasan. El editor de desarrollo fue Melinda D. Bilecki. El gestor del proyecto fue Melissa Leick. El director de marketing fue Curt Reynolds. El supervisor de producción fue Laura Fuller. El diseñador del libro fue Brenda Rolwes. El diseño de la cubierta fue de Studio Montage, St. Louis, Missouri. El editor de copia George Watson. El revisor fue Kevin Campbell. El indizador *freelance* fue Tobiah Waldron. El equipo Aptara lo forman Raman Arora y Sudeshna Nandy.

Notas personales

Sudarshan quiere agradecer a su esposa, Sita, su amor y su apoyo, y a sus hijos Madhur y Advaith por su amor y ganas de vivir. Hank quiere agradecer a su mujer, Joan, y a sus hijos, Abby y Joe, su amor y su comprensión. Avi quiere agradecer a Valerie su amor, su paciencia y su apoyo durante la revisión de este libro.

A. S.

H. F. K.

S. S

Introducción

01

Un **sistema gestor de bases de datos (SGBD)** consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos. La colección de datos, normalmente denominada **base de datos,** contiene información relevante para una empresa. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto *práctica* como *eficiente*.

Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la misma. Además, los sistemas de bases de datos deben garantizar la fiabilidad de la información almacenada, a pesar de las caídas del sistema o de los intentos de acceso no autorizados. Si los datos van a ser compartidos entre diferentes usuarios, el sistema debe evitar posibles resultados anómalos.

Dado que la información es tan importante en la mayoría de las organizaciones, los científicos informáticos han desarrollado un gran cuerpo de conceptos y técnicas para la gestión de los datos. Estos conceptos y técnicas constituyen el objetivo central de este libro. En este capítulo se presenta una breve introducción a los principios de los sistemas de bases de datos.

1.1. Aplicaciones de los sistemas de bases de datos

Las bases de datos se usan ampliamente. Algunas de sus aplicaciones representativas son:

- *Información empresarial*
  - *Ventas:* para información de clientes, productos y compras.
  - *Contabilidad:* para pagos, recibos, contabilidad, asientos y otra información contable.
  - *Recursos humanos:* para información sobre empleados, salarios, pago de impuestos y beneficios, así como para la generación de las nóminas.
  - *Fabricación:* para la gestión de la cadena de suministros y el seguimiento de la producción en fábrica, la gestión de inventarios en los almacenes y la gestión de pedidos.
  - *Comercio en línea:* para los datos de ventas ya mencionados y para el seguimiento de los pedidos web, generación de listas de recomendaciones y mantenimiento de evaluaciones de productos en línea.
- *Banca y finanzas*
  - *Banca:* para información de los clientes, cuentas, préstamos y transacciones bancarias.
  - *Transacciones de tarjetas de crédito:* para compras con tarjeta de crédito y la generación de los extractos mensuales.
  - *Finanzas:* para almacenar información sobre compañías tenedoras; ventas y compras de productos financieros, como acciones y bonos, y también para almacenar datos del mercado en tiempo real que permitan a los clientes la compraventa en línea y a la compañía la compraventa automática.
- *Universidades:* para la información sobre estudiantes, la inscripción en cursos y la graduación (además de la información habitual de recursos humanos y contabilidad).
- *Líneas aéreas:* para reservas e información de horarios. Las líneas aéreas fueron de las primeras en usar las bases de datos de forma distribuida geográficamente.
- *Telecomunicaciones:* para guardar un registro de las llamadas realizadas, generar las facturas mensuales, mantener el saldo de las tarjetas telefónicas de prepago y almacenar información sobre las redes de comunicaciones.

Como se muestra en la lista, en la actualidad las bases de datos son una parte esencial de todas las empresas, donde se almacena no solo información típica de cualquier empresa, sino también la que es específica de cada tipo de empresa.

Durante las últimas cuatro décadas del siglo XX, el uso de las bases de datos creció en todas las empresas. En los primeros días, muy pocas personas interactuaban directamente con los sistemas de bases de datos, aunque sin darse cuenta interactuaban indirectamente con bases de datos (con informes impresos, como los extractos de las tarjetas de crédito, o mediante agentes, como los cajeros de los bancos y los agentes de reservas de las líneas aéreas). Después vinieron los cajeros automáticos y permitieron a los usuarios interactuar directamente con las bases de datos. Las interfaces telefónicas con las computadoras (sistemas de respuesta vocal interactiva) también permitieron a los usuarios tratar directamente con las bases de datos (la persona que llamaba podía marcar un número y pulsar las teclas del teléfono para introducir información o para seleccionar opciones alternativas, para conocer las horas de llegada o salida de los vuelos, por ejemplo, o para matricularse de asignaturas en una universidad).

La revolución de Internet a finales de los años noventa aumentó significativamente el acceso directo del usuario a las bases de datos. Las organizaciones convirtieron muchas de sus interfaces telefónicas a las bases de datos en interfaces web, y dejaron disponibles en línea muchos servicios. Por ejemplo, cuando se accede a una librería en línea y se busca en una colección de libros o de música, se está accediendo a datos almacenados en una base de datos. Cuando se realiza un pedido en línea, el pedido se almacena en una base de datos. Cuando se accede al sitio web de un banco y se consultan el estado de la cuenta y los movimientos, la información se recupera del sistema de bases de datos del banco. Cuando se accede a un sitio web, puede que se recupere información personal de una base de datos para seleccionar los anuncios que se deben mostrar. Más aún, los datos sobre los accesos web pueden almacenarse en una base de datos.

Así, aunque las interfaces de usuario ocultan los detalles del acceso a las bases de datos, y la mayoría de la gente ni siquiera es consciente de que está interactuando con una base de datos, el acceso a las bases de datos forma actualmente una parte esencial de la vida de casi todas las personas.

La importancia de los sistemas de bases de datos se puede juzgar de otra forma; actualmente, los fabricantes de sistemas de bases de datos, como Oracle, están entre las mayores compañías de software del mundo, y los sistemas de bases de datos forman una parte importante de la línea de productos de compañías más diversificadas, como Microsoft e IBM.

1.2. Propósito de los sistemas de bases de datos

Los sistemas de bases de datos surgieron en respuesta a los primeros métodos de gestión informatizada de los datos comerciales. Como ejemplo de este tipo de métodos, típicos de los años sesenta, suponga una parte de una organización, como una universidad, que entre otros datos guarda información sobre profesores, estudiantes, departamentos y oferta de cursos. Una manera de guardar la información en la computadora es almacenarla en archivos del sistema operativo. Para permitir que los usuarios manipulen la información, el sistema tiene varios programas de aplicación que gestionan los archivos, incluyendo programas para:

- Añadir nuevos estudiantes, profesores y cursos.
- Inscribir a los estudiantes en cursos y generar grupos de clases.
- Poner notas a los estudiantes, calcular notas medias y generar certificados.

Estos programas de aplicación los han escrito programadores de sistemas en respuesta a las necesidades de la universidad.

Según se vayan necesitando, se añaden nuevas funciones al sistema. Por ejemplo, suponga que la universidad decide ofertar un nuevo título, digamos que en Informática. La universidad debe crear un nuevo departamento y nuevos archivos permanentes, o añadir información a los archivos existentes, para registrar la información de los profesores del departamento, los estudiantes que se encuentran en dicho título, la oferta de asignaturas, los requisitos de acceso, etc. Puede que se necesite también escribir programas para manejar las restricciones que existan para el nuevo título, así como reglas que imponga la universidad. En este sentido, según pasa el tiempo el sistema va ganando más y más archivos y programas de aplicación.

Los sistemas operativos convencionales soportan este **sistema de procesamiento de archivos** típico. El sistema almacena los registros permanentes en varios archivos y necesita diferentes programas de aplicación para extraer y añadir datos a los archivos correspondientes. Antes de la aparición de los sistemas gestores de bases de datos (SGBD), las organizaciones normalmente almacenaban la información en sistemas de este tipo.

Guardar la información de la organización en un sistema de procesamiento de archivos tiene una serie de inconvenientes importantes:

- **Redundancia e inconsistencia de los datos.** Debido a que los archivos y programas de aplicación los crean diferentes programadores en el transcurso de un largo periodo de tiempo, es probable que los diversos archivos tengan estructuras diferentes y que los programas estén escritos en varios lenguajes de programación diferentes. Además, puede que la información esté duplicada en varios lugares (archivos). Por ejemplo, si un estudiante está matriculado en una titulación doble, por ejemplo Música y Matemáticas, la dirección y el número de teléfono de ese estudiante pueden estar en un archivo con los registros de todos los estudiantes del departamento de Música y en otro archivo con los registros de los estudiantes del departamento de Matemáticas. Esta redundancia genera mayores necesidades de almacenamiento a un mayor coste. Además, puede generar **inconsistencia de datos,** es decir, puede que las distintas copias de los datos no coincidan; por ejemplo, el estudiante cambia su dirección en el registro del departamento de Matemáticas pero no en el resto del sistema.
- **Dificultad en el acceso a los datos.** Suponga que uno de los empleados de la universidad necesita conocer los nombres de todos los estudiantes que viven en un determinado código postal. Este empleado pide al departamento de procesamiento de datos que genere esa lista. Debido a que esta petición no fue prevista por los diseñadores del sistema, no hay un programa de aplicación para satisfacerla. Hay, sin embargo, un programa de aplicación que genera la lista de todos los estudiantes. El empleado tiene dos opciones: bien obtener la lista de *todos* los estudiantes y extraer manualmente la información que necesita, o bien pedir a un programador que escriba el programa de aplicación necesario. Ambas alternativas son obviamente poco satisfactorias. Suponga que se escribe el programa y que, varios días más tarde, el mismo empleado necesita reducir esa lista para que incluya únicamente a aquellos estudiantes matriculados al menos en 60 créditos. Como cabría esperar, no existe ningún programa que genere tal lista. De nuevo, el empleado tiene que elegir entre dos opciones, ninguna de las cuales es satisfactoria.

  La cuestión entonces es que los entornos de procesamiento de archivos convencionales no permiten recuperar los datos necesarios de una forma práctica y eficiente. Hacen falta sistemas de recuperación de datos más adecuados para el uso general.
- **Aislamiento de datos.** Como los datos están dispersos en varios archivos, y los archivos pueden estar en diferentes formatos, es difícil escribir nuevos programas de aplicación para recuperar los datos correspondientes.
- **Problemas de integridad.** Los valores de los datos almacenados en la base de datos deben satisfacer ciertos tipos de **restricciones de consistencia.** Suponga que la universidad mantiene una cuenta para cada departamento y registros de los saldos de dicha cuenta. Suponga también, que la universidad requiere que el saldo de un departamento nunca sea menor que cero. Los desarrolladores hacen cumplir esas restricciones en el sistema añadiendo el código correspondiente en los diversos programas de aplicación. Sin embargo, cuando se añaden nuevas restricciones, es difícil cambiar los programas para hacer que se cumplan. El problema se complica cuando las restricciones implican diferentes elementos de datos de diversos archivos.
- **Problemas de atomicidad.** Los sistemas informáticos, como cualquier otro dispositivo mecánico o eléctrico, están sujetos a fallos. En muchas aplicaciones es crucial asegurar que, si se produce algún fallo, los datos se restauren al estado consistente que existía antes del fallo. Suponga un programa para transferir 500 € de la cuenta de un departamento A a la cuenta de un departamento B. Si se produce un fallo en el sistema durante la ejecución del programa, es posible que se hayan sacado 500 € de la cuenta del departamento A pero todavía no se hayan ingresado en el departamento B, generando un estado inconsistente de la base de datos. Evidentemente, para la consistencia de la base de datos resulta esencial que tengan lugar tanto el abono como el cargo, o que no tenga lugar ninguno. Es decir, la transferencia de fondos debe ser *atómica*; debe ocurrir en su totalidad o no ocurrir en absoluto. Resulta difícil asegurar la atomicidad en los sistemas convencionales de procesamiento de archivos.

- **Anomalías en el acceso concurrente.** Para aumentar el rendimiento global del sistema y obtener una respuesta más rápida, muchos sistemas permiten que varios usuarios actualicen los datos simultáneamente. En realidad, hoy en día los principales sitios de comercio electrónico de Internet pueden tener millones de accesos diarios de compradores a sus datos. En tales entornos es posible la interacción de actualizaciones concurrentes que pueden dar lugar a datos inconsistentes. Suponga que un departamento A tiene una cuenta con un saldo de 10.000 €. Si dos empleados retiran fondos (por ejemplo, 500 € y 100 €, respectivamente) de la cuenta *A* exactamente a la vez, el resultado de las ejecuciones concurrentes puede dejar la cuenta en un estado incorrecto, o inconsistente. Suponga que los programas que se ejecutan para cada retirada leen el saldo anterior, reducen su valor en el importe que se retira y luego escriben el resultado. Si los dos programas se ejecutan concurrentemente, pueden leer el valor 10.000 €, y escribir después 9.500 € y 9.900 €, respectivamente. Dependiendo de cuál escriba el valor en último lugar, la cuenta puede contener 9.500 € o 9.900 €, en lugar del valor correcto de 9.400 €. Para protegerse contra esta posibilidad, el sistema debe mantener alguna forma de supervisión. Pero es difícil ofrecer supervisión, ya que pueden tener acceso a los datos muchos programas de aplicación diferentes que no se hayan coordinado con anterioridad.

  Otro ejemplo: suponga que un programa de matriculación mantiene la cuenta de estudiantes matriculados en una asignatura, para restringir el número de estudiantes que se pueden matricular. Cuando un estudiante se matricula, el programa lee la cuenta actual, verifica que la cuenta no ha superado el límite, añade 1 a la cuenta y la guarda en la base de datos. Suponga que dos estudiantes se matriculan concurrentemente cuando la cuenta vale, por ejemplo, 39. Las dos ejecuciones del programa pueden leer el valor de cuenta de 39, y ambos escribirán el valor de 40, lo que supone incrementar de forma incorrecta el valor en 1, aunque realmente se hayan registrado efectivamente dos estudiantes y la cuenta debería ser de 41. Más aún, suponga que el límite de matriculación en el curso fuese de 40; en el caso anterior ambos estudiantes se habrían conseguido matricular violando el límite de los 40 estudiantes.
- **Problemas de seguridad.** Ningún usuario del sistema de la base de datos debería poder acceder a todos los datos. Por ejemplo, en una universidad, el personal del departamento financiero debería poder acceder exclusivamente a la parte de la base de datos con información financiera. No necesitan acceder a la información de registros académicos. Como los programas de aplicación se van añadiendo al sistema según se necesitan, resulta difícil forzar este tipo de restricciones de seguridad.

Estas dificultades, entre otras, han motivado el desarrollo de los sistemas de bases de datos. En el resto del libro se examinarán los conceptos y algoritmos que permiten que los sistemas de bases de datos resuelvan los problemas de los sistemas de procesamiento de archivos. En la mayor parte del libro se usa una universidad como ejemplo de aplicación típica de procesamiento de datos.

1.3. Visión de los datos

Un sistema de base de datos es una colección de datos interrelacionados y un conjunto de programas que permiten a los usuarios tener acceso a esos datos y modificarlos. Una de las principales finalidades de los sistemas de bases de datos es ofrecer a los usuarios una visión *abstracta* de los datos. Es decir, el sistema oculta ciertos detalles del modo en que se almacenan y mantienen los datos.

1.3.1. Abstracción de datos

Para que el sistema sea útil debe recuperar los datos eficientemente. La necesidad de eficiencia ha llevado a los diseñadores a usar estructuras de datos complejas para la representación de los datos en la base de datos. Dado que muchos de los usuarios de sistemas de bases de datos no tienen formación en informática, los desarrolladores ocultan esa complejidad a los usuarios mediante varios niveles de abstracción para simplificar la interacción de los usuarios con el sistema:

- **Nivel físico.** El nivel más bajo de abstracción describe *cómo* se almacenan realmente los datos. El nivel físico describe en detalle las estructuras de datos complejas de bajo nivel.
- **Nivel lógico.** El nivel inmediatamente superior de abstracción describe *qué* datos se almacenan en la base de datos y qué relaciones existen entre esos datos. El nivel lógico, por tanto, describe toda la base de datos en términos de un número pequeño de estructuras relativamente simples. Aunque la implementación de esas estructuras simples en el nivel lógico puede involucrar estructuras complejas del nivel físico, los usuarios del nivel lógico no necesitan preocuparse de esta complejidad. A esto se denomina **independencia de los datos físicos.** Los administradores de bases de datos, que deben decidir la información que se guarda en la base de datos, usan el nivel de abstracción lógico.
- **Nivel de vistas.** El nivel más elevado de abstracción solo describe parte de la base de datos. Aunque el nivel lógico usa estructuras más simples, queda cierta complejidad debido a la variedad de información almacenada en las grandes bases de datos. Muchos usuarios del sistema de bases de datos no necesitan toda esta información; en su lugar solo necesitan tener acceso a una parte de la base de datos. El nivel de abstracción de vistas existe para simplificar su interacción con el sistema. El sistema puede proporcionar muchas vistas para la misma base de datos.

En la Figura 1.1 se muestra la relación entre los tres niveles de abstracción.

Una analogía con el concepto de tipos de datos en lenguajes de programación puede clarificar la diferencia entre los niveles de abstracción. La mayoría de los lenguajes de programación de alto nivel soportan el concepto de tipo estructurado. Por ejemplo, en los lenguajes tipo Pascal se pueden declarar registros de la manera siguiente[1]:

type *profesor* = **record**
  ID: **char** (5);
  nombre: **char** (20);
  nombre_dept: **char** (20);
  sueldo: **numeric** (8,2);
end;

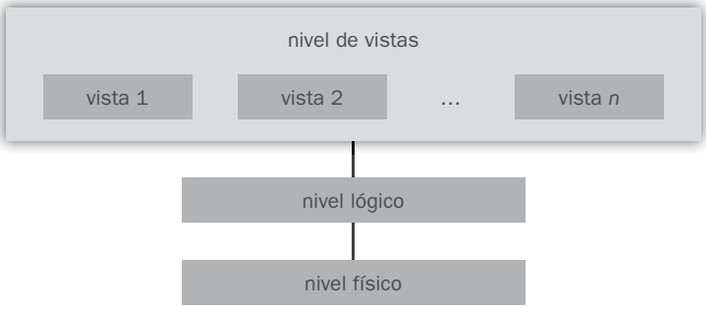


Figura 1.1. Los tres niveles de abstracción de datos.

1 La declaración de tipos real dependerá del lenguaje que se utilice. En C y C++ se utiliza la declaración **struct.** En Java no existe este tipo de declaración, sino que se obtiene el mismo efecto definiendo una clase simple.

Este código define un nuevo tipo de registro denominado *profesor*, con cuatro campos. Cada campo tiene un nombre y un tipo asociados. Una organización como una universidad puede tener muchos tipos de registros, entre otros:

- *departamento*, con los campos *nombre_dept, edificio* y *presupuesto*.
- *asignatura*, con los campos *id_asignatura, nombre_asig, nombre_dept* y *créditos*.
- *estudiante*, con los campos *nombre, nombre_dept* y *total_créditos*.

En el nivel físico, los registros profesor, departamento o estudiante se pueden describir como bloques de posiciones consecutivas de almacenamiento. El compilador oculta este nivel de detalle a los programadores. De manera parecida, el sistema de base de datos oculta muchos de los detalles de almacenamiento de los niveles inferiores a los programadores de bases de datos. Los administradores de bases de datos, por otro lado, pueden ser conscientes de ciertos detalles de la organización física de los datos.

En el nivel lógico, cada registro de este tipo se describe mediante una definición de tipo, como en el fragmento de código anterior, y también se define la relación entre estos tipos de registros. Los programadores que usan un lenguaje de programación trabajan en este nivel de abstracción. De manera parecida, los administradores de bases de datos suelen trabajar en este nivel de abstracción.

Finalmente, en el nivel de vistas, los usuarios ven un conjunto de programas de aplicación que ocultan los detalles de los tipos de datos. En el nivel de vistas se definen varias vistas de la base de datos, y los usuarios de estas pueden verlas todas o solamente una parte. Además de ocultar los detalles del nivel lógico de la base de datos, las vistas también proporcionan un mecanismo de seguridad para evitar que los usuarios tengan acceso a ciertas partes. Por ejemplo, los empleados de matriculación de la universidad solamente pueden ver la parte sobre información de los estudiantes, pero no pueden acceder a la información sobre sueldos ni profesores.

1.3.2. Ejemplares y esquemas

Las bases de datos van cambiando a lo largo del tiempo conforme la información se inserta y se elimina. La colección de información almacenada en la base de datos en un momento dado se denomina **ejemplar** de la base de datos. El diseño general de la base de datos se denomina **esquema** de la base de datos. Los esquemas se modifican rara vez, si es que se modifican.

El concepto de esquemas y ejemplares de las bases de datos se puede comprender por analogía con los programas escritos en un lenguaje de programación. El esquema de la base de datos se corresponde con las declaraciones de las variables (junto con las definiciones de tipos asociadas) de los programas. Cada variable tiene un valor concreto en un instante dado. Los valores de las variables de un programa se corresponden en cada momento con un *ejemplar* del esquema de la base de datos.

Los sistemas de bases de datos tienen varios esquemas, divididos según los niveles de abstracción. El **esquema físico** describe el diseño de la base de datos en el nivel físico, mientras que el **esquema lógico** describe su diseño en el nivel lógico. Las bases de datos también pueden tener varios esquemas en el nivel de vistas, a veces denominados **subesquemas,** que describen diferentes vistas de la base de datos.

De estos, el esquema lógico es con mucho el más importante en términos de su efecto sobre los programas de aplicación, ya que los programadores crean las aplicaciones usando el esquema lógico. El esquema físico está oculto bajo el esquema lógico, y generalmente puede modificarse fácilmente sin afectar a los programas de aplicación. Se dice que los programas de aplicación muestran **independencia física respecto de los datos** si no dependen del esquema físico y, por tanto, no hace falta volver a escribirlos si se modifica el esquema físico.

Se estudiarán los lenguajes para la descripción de los esquemas, después de introducir el concepto de modelos de datos en la siguiente sección.

1.3.3. Modelos de datos

Bajo la estructura de las bases de datos se encuentra el **modelo de datos:** se trata de una colección de herramientas conceptuales para describir los datos, sus relaciones, su semántica y las restricciones de consistencia. Los modelos de datos ofrecen un modo de describir el diseño de las bases de datos en los niveles físico, lógico y de vistas.

En este texto se van a tratar varios modelos de datos diferentes. Los modelos de datos pueden clasificarse en cuatro categorías diferentes:

- **Modelo relacional.** El modelo relacional utiliza una colección de tablas para representar tanto los datos como sus relaciones. Cada tabla tiene varias columnas, y cada columna tiene un nombre único. Las tablas también son conocidas como **relaciones.** El modelo relacional es un ejemplo de modelo basado en registros. Los modelos basados en registros se denominan así porque la base de datos se estructura en registros de formato fijo de varios tipos. Cada tabla contiene registros de un tipo dado. Cada tipo de registro define un número fijo de campos, o atributos. Las columnas de la tabla se corresponden con los atributos del tipo de registro. El modelo de datos relacional es el modelo de datos más ampliamente utilizado, y una gran mayoría de sistemas de bases de datos actuales se basan en el modelo relacional. Los Capítulos 2 al 8 tratan el modelo relacional en detalle.
- **El modelo entidad-relación.** El modelo de datos entidad-relación (E-R) consiste en una colección de objetos básicos, denominados *entidades*, y de las *relaciones* entre ellos. Una entidad es una «cosa» u «objeto» del mundo real que es distinguible de otros objetos. El modelo entidad-relación se usa mucho en el diseño de bases de datos, y en el Capítulo 7 se examina detalladamente.
- **Modelo de datos basado en objetos.** La programación orientada a objetos, especialmente en Java, C++ o C#, se ha convertido en la metodología de desarrollo software dominante, lo que ha llevado al desarrollo de modelos de datos orientados a objetos. El modelo orientado a objetos se puede considerar como una extensión del modelo E-R con conceptos de encapsulación, métodos (funciones) e identidad de objetos. El modelo de datos relacional de objetos combina las características de los modelos de datos orientados a objetos y el modelo de datos relacional. En el Capítulo 22 se examina este modelo de datos.
- **Modelo de datos semiestructurados.** El modelo de datos semiestructurados permite la especificación de datos en el que los elementos de datos individuales del mismo tipo pueden tener diferentes conjuntos de atributos. Esto lo diferencia de los modelos mencionados anteriormente, en los que cada elemento de datos de un tipo particular debe tener el mismo conjunto de atributos. El **lenguaje de marcas extensible** (XML: *eXtensible Markup Language*) se emplea mucho para representar datos semiestructurados. Se estudia en el Capítulo 23.

El **modelo de datos de red** y el **modelo de datos jerárquico** precedieron cronológicamente al relacional. Estos modelos estuvieron íntimamente ligados a la implementación subyacente y complicaban la tarea del modelado de datos. En consecuencia, se usan muy poco hoy en día, excepto en el código de bases de datos antiguas que siguen estando en servicio en algunos lugares. Se describen brevemente en los Apéndices D y E para los lectores interesados.

1.4. Lenguajes de bases de datos

Los sistemas de bases de datos proporcionan un **lenguaje de definición de datos** para especificar el esquema de la base de datos, y un **lenguaje de manipulación de datos** para expresar las consultas y las modificaciones de la base de datos. En la práctica, los lenguajes de definición y manipulación de datos no son dos lenguajes diferentes; simplemente forman parte de un único lenguaje de bases de datos, como puede ser el muy usado SQL.

1.4.1. Lenguaje de manipulación de datos

Un **lenguaje de manipulación de datos (LMD)** es un lenguaje que permite a los usuarios tener acceso a los datos organizados mediante el modelo de datos correspondiente, o manipularlos. Los tipos de acceso son:

- La recuperación de la información almacenada en la base de datos.
- La inserción de información nueva en la base de datos.
- El borrado de la información de la base de datos.
- La modificación de la información almacenada en la base de datos.

Hay fundamentalmente dos tipos:

- Los **LMD procedimentales** necesitan que el usuario especifique *qué* datos se necesitan y *cómo* obtener esos datos.
- Los **LMD declarativos** (también conocidos como **LMD no procedimentales**) necesitan que el usuario especifique *qué* datos se necesitan *sin* que haga falta especificar cómo obtener esos datos.

Los LMD declarativos suelen resultar más fáciles de aprender y de usar que los procedimentales. Sin embargo, como el usuario no tiene que especificar cómo conseguir los datos, el sistema de bases de datos tiene que determinar un medio eficiente de acceso a los mismos.

Una **consulta** es una instrucción que solicita recuperar información. La parte de los LMD implicada en la recuperación de información se denomina **lenguaje de consultas.** Aunque técnicamente sea incorrecto, resulta habitual usar las expresiones *lenguaje de consultas* y *lenguaje de manipulación de datos* como sinónimas.

Existen varios lenguajes de consultas de bases de datos en uso, tanto comercial como experimental. El lenguaje de consultas más ampliamente usado, SQL, se estudiará en los Capítulos 3, 4 y 5. También se estudiarán otros lenguajes de consultas en el Capítulo 6.

Los niveles de abstracción que se trataron en la Sección 1.3 no solo se aplican a la definición o estructuración de datos, sino también a su manipulación. En el nivel físico se deben definir los algoritmos que permitan un acceso eficiente a los datos. En los niveles superiores de abstracción el énfasis se pone en la facilidad de uso. El objetivo es permitir que los seres humanos interactúen de manera eficiente con el sistema. El componente procesador de consultas del sistema de bases de datos (que se estudia en los Capítulos 12 y 13) traduce las consultas LMD en secuencias de acciones en el nivel físico del sistema de bases de datos.

1.4.2. Lenguaje de definición de datos

Los esquemas de las bases de datos se especifican mediante un conjunto de definiciones expresadas mediante un lenguaje especial denominado **lenguaje de definición de datos (LDD).** El LDD también se usa para especificar más propiedades de los datos.

La estructura de almacenamiento y los métodos de acceso usados por el sistema de bases de datos se especifican mediante un conjunto de instrucciones en un tipo especial de LDD denominado **lenguaje de almacenamiento y definición de datos.** Estas instrucciones definen los detalles de implementación de los esquemas de las bases de datos, que suelen ocultarse a los usuarios.

Los valores de los datos almacenados en la base de datos deben satisfacer ciertas **restricciones de consistencia.** Por ejemplo, suponga que la universidad requiere que el saldo de una cuenta de un departamento nunca pueda ser un valor negativo. El LDD proporciona elementos para especificar tales restricciones. Los sistemas de bases de datos los comprueban cada vez que se modifica la base de datos. En general, las restricciones pueden ser predicados arbitrarios relativos a la base de datos. No obstante, los predicados arbitrarios pueden resultar costosos de comprobar. Por tanto, los sistemas de bases de datos se concentran en las restricciones de integridad que pueden comprobarse con una sobrecarga mínima:

- **Restricciones de dominio.** Se debe asociar un dominio de valores posibles a cada atributo (por ejemplo, tipos enteros, tipos de carácter, tipos fecha/hora). La declaración de un atributo como parte de un dominio concreto actúa como restricción de los valores que puede adoptar. Las restricciones de dominio son la forma más elemental de restricción de integridad. El sistema las comprueba fácilmente siempre que se introduce un nuevo elemento de datos en la base de datos.
- **Integridad referencial.** Hay casos en los que se desea asegurar que un valor que aparece en una relación para un conjunto de atributos dado aparezca también para un determinado conjunto de atributos en otra relación (integridad referencial). Por ejemplo, el departamento asociado a cada asignatura debe existir realmente. De forma más precisa, el valor *nombre_dept* de un registro de *asignatura* tiene que aparecer en el atributo *nombre_dept* de algún registro de la relación *departamento*. Las modificaciones de la base de datos pueden causar violaciones de la integridad referencial. Cuando se viola una restricción de integridad, el procedimiento normal es rechazar la acción que ha causado esa violación.
- **Asertos.** Un aserto es cualquier condición que la base de datos debe satisfacer siempre. Las restricciones de dominio y las restricciones de integridad referencial son formas especiales de asertos. No obstante, hay muchas restricciones que no pueden expresarse empleando únicamente esas formas especiales. Por ejemplo, «todos los departamentos deben ofertar al menos cinco asignaturas cada semestre» debe expresarse en forma de aserto. Cuando se crea un aserto, el sistema comprueba su validez. Si el aserto es válido, cualquier modificación futura de la base de datos únicamente se permite si no hace que se viole ese aserto.
- **Autorización.** Puede que se desee diferenciar entre los usuarios en cuanto al tipo de acceso que se les permite a los diferentes valores de los datos de la base de datos. Estas diferenciaciones se expresan en términos de **autorización,** cuyas modalidades más frecuentes son: **autorización de lectura,** que permite la lectura pero no la modificación de los datos; **autorización de inserción,** que permite la inserción de datos nuevos pero no la modificación de los datos ya existentes; **autorización de actualización,** que permite la modificación pero no la eliminación de los datos; y la **autorización de eliminación,** que permite la eliminación de datos. A cada usuario se le pueden asignar todos, ninguno, o una combinación de estos tipos de autorización.

El LDD, al igual que cualquier otro lenguaje de programación, obtiene como entrada algunas instrucciones (sentencias) y genera una salida. La salida del LDD se coloca en el **diccionario de datos,** que contiene **metadatos,** es decir, datos sobre datos. El diccionario de datos se considera un tipo especial de tabla, solo accesible y actualizable por el propio sistema de bases de datos (no por los usuarios normales). El sistema de bases de datos consulta el diccionario de datos antes de leer o modificar los datos reales.

1.5. Bases de datos relacionales

Las bases de datos relacionales se basan en el modelo relacional y usan un conjunto de tablas para representar tanto los datos como las relaciones entre ellos. También incluyen un LMD y un LDD. En el Capítulo 2 se presenta una introducción a los fundamentos del modelo relacional. La mayor parte de los sistemas de bases de datos relacionales comerciales emplean el lenguaje SQL, que se tratará con gran detalle en los Capítulos 3, 4 y 5. En el Capítulo 6 se estudiarán otros lenguajes influyentes.

1.5.1. Tablas

Cada tabla tiene varias columnas, y cada columna tiene un nombre único. En la Figura 1.2 se presenta un ejemplo de base de datos relacional consistente en dos tablas: una muestra detalles de los profesores, la otra muestra los distintos departamentos.

La primera tabla, la tabla *profesor*, muestra, por ejemplo, que el profesor de nombre Einstein, cuyo *ID* es 22222, forma parte del departamento de Física y tiene un sueldo anual de 95.000 €. La segunda tabla, de *departamento*, muestra, por ejemplo, que el departamento de Biología está ubicado en el edificio Watson y tiene un presupuesto de 95.000 €. Por supuesto, una universidad real tendrá muchos más departamentos y profesores. En el texto se usarán tablas pequeñas para ilustrar los conceptos. Se encuentra disponible *online* un ejemplo mucho mayor del mismo esquema.

El modelo relacional es un ejemplo de modelo basado en registros. Los modelos basados en registros se denominan así porque la base de datos se estructura en registros de formato fijo de varios tipos. Cada tabla contiene registros de un tipo dado. Cada tipo de registro define un número fijo de campos, o atributos. Las columnas de la tabla se corresponden con los atributos del tipo de registro.

ID	*nombre*	*nombre_dept*	*sueldo*
22222	Einstein	Física	95.000
12121	Wu	Finanzas	90.000
32343	El Said	Historia	60.000
45565	Kafz	Informática	75.000
98345	Kim	Electrónica	80.000
76766	Crick	Biología	72.000
10101	Srinivasan	Informática	65.000
58583	Califieri	Historia	62.000
83821	Brandt	Informática	92.000
15151	Mozart	Música	40.000
33456	Gold	Física	87.000
76543	Singh	Finanzas	80.000

(a) La tabla *profesor*.

nombre_dept	*edificio*	*presupuesto*
Informática	Taylor	100.000
Biología	Watson	90.000
Electrónica	Taylor	85.000
Música	Packard	80.000
Finanzas	Painter	120.000
Historia	Painter	50.000
Física	Watson	70.000

(b) La tabla *departamento*.

Figura 1.2. Un ejemplo de base de datos relacional.

No es difícil ver cómo se pueden almacenar las tablas en archivos. Por ejemplo, se puede usar un carácter especial (como la coma) para delimitar los diferentes atributos de un registro, y otro carácter especial (como el carácter de nueva línea) para delimitar los registros. El modelo relacional oculta esos detalles de implementación de bajo nivel a los desarrolladores de bases de datos y a los usuarios.

Obsérvese también que en el modelo relacional es posible crear esquemas que tengan problemas tales como información duplicada innecesariamente. Por ejemplo, supóngase que se almacena el *presupuesto* del departamento como atributo del registro *profesor*. Entonces, siempre que cambie el valor de un determinado presupuesto, por ejemplo el del departamento de Física, este cambio debe reflejarse en los registros de todos los profesores asociados con el departamento de Física. En el Capítulo 8 se tratará cómo distinguir los buenos diseños de esquema de los malos.

1.5.2. Lenguaje de manipulación de datos

El lenguaje de consultas de SQL no es procedimental. Usa como entrada varias tablas (posiblemente solo una) y devuelve siempre una única tabla. A continuación se ofrece un ejemplo de consulta SQL que halla el nombre de todos los profesores del departamento de Historia:

select *profesor.nombre*
from *profesor*
where *profesor.nombre_dept* = «Historia»

La consulta especifica que hay que recuperar *(select)* las filas de *(from)* la tabla *profesor* en las que *(where)* el *nombre_dept* es Historia, y que solo debe mostrarse el atributo *nombre* de esas filas. Más concretamente, el resultado de la ejecución de esta consulta es una tabla con una sola columna denominada *nombre* y un conjunto de filas, cada una de las cuales contiene el nombre de un profesor cuyo *nombre_dept* es Historia. Si la consulta se ejecuta sobre la tabla de la Figura 1.2, el resultado constará de dos filas, una con el nombre El Said y otra con el nombre Califieri.

Las consultas pueden requerir información de más de una tabla. Por ejemplo, la siguiente consulta busca todos los *ID* de profesor y el nombre del departamento de todos los profesores con un departamento cuyo presupuesto sea superior a 95.000 €.

select *profesor.ID*, *departamento.nombre_dept*
from *profesor*, *departamento*
where *profesor.nombre_dept* = *departamento.nombre_dept* **and**
departamento.presupuesto > 95000;

Si la consulta anterior se ejecutase sobre las tablas de la Figura 1.2, el sistema encontraría que hay dos departamentos con un presupuesto superior a 95.000 €, Informática y Finanzas, y que hay cinco profesores en estos departamentos. Por tanto, el resultado consistiría en una tabla con dos columnas *(ID, nombre_dept)* y las cinco filas: (12121, Finanzas), (45565, Informática), (10101, Informática), (83821, Informática) y (76543, Finanzas).

1.5.3. Lenguaje de definición de datos

SQL proporciona un rico LDD que permite definir tablas, restricciones de integridad, aserciones, etc.

Por ejemplo, la siguiente instrucción LDD del lenguaje SQL define la tabla *departamento*:

create table *departamento*
(*nombre_dept* **char** (20),
edificio **char** (15),
presupuesto **numeric** (12,2));

La ejecución de esta instrucción LDD crea la tabla *departamento* con tres columnas: *nombre_dept*, *edificio* y *presupuesto*, cada una de ellas con su tipo de datos específico asociado. Se tratarán los tipos de datos con más detalle en el Capítulo 3. Además, la instrucción LDD actualiza el diccionario de datos, que contiene metadatos (véase la Sección 1.4.2). El esquema de la tabla es un ejemplo de metadatos.

1.5.4. Acceso a las bases de datos desde los programas de aplicación

SQL no es tan potente como una máquina universal de Turing; es decir, hay algunos cálculos que se pueden conseguir mediante un lenguaje de programación general pero no se pueden obtener mediante consultas SQL. SQL tampoco permite acciones como la entrada de datos de usuario, mostrar datos en pantalla o la comunicación por la red. Estas operaciones se deben escribir en otro lenguaje *anfitrión*, como por ejemplo C, C++ o Java, capaces de albergar consultas SQL que acceden a los datos de la base de datos. Los **programas de aplicación** son programas que se usan para interactuar de esta manera con las bases de datos. Algunos ejemplos de un sistema universitario serían los programas que permiten a los estudiantes matricularse en un curso, generar horarios, calcular las notas medias de los estudiantes, generar las nóminas, etc.

Para tener acceso a la base de datos, las instrucciones LMD deben ejecutarse desde el lenguaje anfitrión. Hay dos maneras de conseguirlo:

- Proporcionando una interfaz de programas de aplicación (conjunto de procedimientos) que se pueda usar para enviar instrucciones LMD y LDD a la base de datos y recuperar los resultados. El estándar de conectividad abierta de bases de datos (ODBC: *Open Data Base Connectivity*) para su empleo con el lenguaje C es un estándar de interfaz de programas de aplicación usado habitualmente. El estándar de conectividad de Java con bases de datos (JDBC: *Java Data Base Connectivity*) ofrece las características correspondientes para el lenguaje Java.
- Extendiendo la sintaxis del lenguaje anfitrión para que incorpore las llamadas LMD dentro del programa del lenguaje anfitrión. Generalmente, un carácter especial precede a las llamadas LMD y un preprocesador, denominado **precompilador LMD,** convierte las instrucciones LMD en llamadas normales a procedimientos en el lenguaje anfitrión.

1.6. Diseño de bases de datos

Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. Esas grandes cantidades de información no existen aisladas. Forman parte del funcionamiento de alguna empresa, cuyo producto final puede que sea la información obtenida de la base de datos o algún dispositivo o servicio para el que la base de datos solo desempeña un papel secundario.

El diseño de bases de datos implica principalmente el diseño de su esquema. El diseño de un entorno completo de aplicaciones para la base de datos que satisfaga las necesidades de la empresa que se está modelando exige prestar atención a un conjunto de aspectos más amplio. Este texto se centrará inicialmente en la escritura de las consultas a una base de datos y en el diseño de los esquemas de bases de datos. En el Capítulo 9 se estudia el proceso general de diseño de las aplicaciones.

1.6.1. Proceso de diseño

Los modelos de datos de alto nivel resultan útiles a los diseñadores de bases de datos al ofrecerles un marco conceptual en el que especificar, de manera sistemática, los requisitos de datos de los usuarios de las bases de datos y la manera en que se estructurará la base de datos para satisfacer esos requisitos. La fase inicial del diseño de las bases de datos, por tanto, consiste en caracterizar completamente los requisitos de datos de los hipotéticos usuarios de la base de datos. Los diseñadores de bases de datos deben interactuar ampliamente con los expertos y usuarios del dominio para llevar a cabo esta tarea. El resultado de esta fase es la especificación de los requisitos de los usuarios.

A continuación, el diseñador escoge un modelo de datos y, mediante la aplicación de los conceptos del modelo de datos elegido, traduce esos requisitos en un esquema conceptual de la base de datos. El esquema desarrollado en esta fase de **diseño conceptual** ofrece una visión general detallada de la empresa. El diseñador revisa el esquema para confirmar que todos los requisitos de datos se satisfacen realmente y no entran en conflicto entre sí. El diseñador también puede examinar el diseño para eliminar cualquier característica redundante. En este punto, la atención se centra en describir los datos y sus relaciones, más que en especificar los detalles del almacenamiento físico.

En términos del modelo relacional, el proceso de diseño conceptual implica decisiones sobre *qué* atributos se desea capturar en la base de datos y *cómo agruparlos* para formar las diferentes tablas. La parte del «qué» es, en esencia, una decisión conceptual, y no se seguirá estudiando en este texto. La parte del «cómo» es sobre todo, un problema informático. Hay dos vías principales para afrontar el problema. La primera supone usar el modelo entidad-relación (Sección 1.6.3); la otra es emplear un conjunto de algoritmos (denominados colectivamente como *normalización*) que toma como entrada el conjunto de todos los atributos y genera un conjunto de tablas (Sección 1.6.4).

Un esquema conceptual completamente desarrollado también indica los requisitos funcionales de la empresa. En la **especificación de requisitos funcionales,** los usuarios describen el tipo de operaciones (o transacciones) que se llevarán a cabo con los datos. Un ejemplo de estas operaciones es modificar o actualizar los datos, buscar y recuperar datos concretos y eliminar datos. En esta etapa del diseño conceptual, el diseñador puede revisar el esquema para asegurarse de que satisface los requisitos funcionales.

El proceso de pasar de un modelo de datos abstracto a la implementación de la base de datos continúa con dos fases de diseño finales. En la **fase de diseño lógico,** el diseñador relaciona el esquema conceptual de alto nivel con el modelo de implementación de datos del sistema de bases de datos que se va a usar. El diseñador usa el esquema de bases de datos específico para el sistema resultante en la **fase de diseño físico** posterior, en la que se especifican las características físicas de la base de datos. Entre esas características están la forma de organización de los archivos y las estructuras de almacenamiento interno; se estudian en el Capítulo 10.

1.6.2. Diseño de la base de datos para una universidad

Para ilustrar el proceso de diseño, vamos a ver cómo se puede diseñar una base de datos para una universidad. La especificación de requisitos inicial puede basarse en un conjunto de entrevistas con los usuarios de la base de datos y el propio análisis del diseñador sobre la organización. La descripción que se obtiene de esta fase de diseño sirve como punto inicial para la especificación de la estructura conceptual de la base de datos. Estas son las principales características de la universidad:

- La universidad está organizada en departamentos. Cada departamento estará identificado por un nombre único *(nombre_dept)*, estará ubicado en un *edificio* y tendrá un *presupuesto*.

- Cada departamento tiene su propia lista de las asignaturas que oferta. Cada asignatura tiene asociado un *asignatura_id*, *nombre_asig*, *nombre_dept*, y puede tener un conjunto de *prerrequisitos*.
- Los profesores estarán identificados por su propio *ID* único. Los profesores tienen un *nombre*, el departamento *(nombre_dept)* al que están asociados y un *sueldo*.
- Los estudiantes estarán identificados por su propio *ID* único. Los estudiantes tendrán un *nombre*, un departamento *(nombre_dept)* principal al que estarán asociados y un *tot_cred* (número total de créditos que el estudiante haya conseguido).
- La universidad mantiene una lista de las aulas, indicando el nombre del *edificio*, el *número_aula* y la *capacidad* del aula.
- La universidad mantiene una lista de las clases (secciones) que se enseñan. Cada sección se identifica por su *asignatura_id*, *secc_id*, *año* y *semestre*, y tiene asociado un *semestre*, *año*, *edificio*, *número_aula* y *franja_horaria_id* (las horas a las que se imparte la clase).
- El departamento tiene una lista de la formación asignada, especificando, para cada profesor, las secciones que enseña.
- La universidad tiene una lista de todos los estudiantes matriculados, especificando, para cada estudiante, las asignaturas y las secciones asociadas en las que se ha matriculado.

La base de datos de una universidad real sería mucho más compleja que la que se ha indicado. Sin embargo, este modelo simplificado nos permitirá entender las ideas conceptuales sin perdernos en los detalles de un diseño complejo.

1.6.3. El modelo entidad-relación

El modelo de datos entidad-relación (E-R) se basa en un conjunto de objetos básicos, denominados *entidades*, y las *relaciones* entre esos objetos. Una entidad es una «cosa» u «objeto» del mundo real que es distinguible de otros objetos. Por ejemplo, cada persona es una entidad, y los profesores pueden considerarse entidades.

Las entidades se describen en las bases de datos mediante un conjunto de **atributos.** Por ejemplo, los atributos *nombre_dept*, *edificio* y *presupuesto* pueden describir un departamento concreto de una universidad y constituyen atributos del conjunto de entidades *departamento*. Análogamente, los atributos *ID*, *nombre* y *sueldo* pueden describir una entidad *profesor*[2].

Se usa un atributo extra, *ID*, para identificar unívocamente a un profesor (dado que es posible que haya dos profesores con el mismo nombre y el mismo sueldo). Se debe asignar un identificador de profesor único a cada profesor. En los Estados Unidos, muchas empresas usan el número de la seguridad social de cada persona (un número único que el Gobierno de Estados Unidos asigna a cada persona) como identificador.

Una **relación** es una asociación entre varias entidades. Por ejemplo, la relación *miembro* asocia un profesor con su departamento. El conjunto de todas las entidades del mismo tipo y el conjunto de todas las relaciones del mismo tipo se denominan, respectivamente, **conjunto de entidades** y **conjunto de relaciones.**

La estructura lógica general (esquema) de la base de datos se puede expresar gráficamente mediante un *diagrama entidad-relación (E-R)*. Existen distintas formas de dibujar este tipo de diagramas. Una de las más populares es utilizar el **lenguaje de modelado unificado** (UML: *unified modeling language*). En la notación que usaremos, basada en UML, un diagrama E-R se representa de la siguiente forma:

- Las entidades se representan mediante un rectángulo con el nombre de la entidad en la cabecera y la lista de atributos debajo.
- Las relaciones se representan mediante un rombo que conecta un par de entidades. El nombre de la relación se pone dentro del rombo.

Como ejemplo, suponga parte de la base de datos de una universidad, que consiste en profesores y departamentos con los que están asociados. En la Figura 1.3 se muestra el diagrama E-R correspondiente. Este diagrama indica que existen dos conjuntos de entidades, profesor y departamento, con los atributos indicados como se ha comentado anteriormente. El diagrama también muestra una relación entre profesor y departamento.

Además de entidades y relaciones, el modelo E-R representa ciertas restricciones que los contenidos de la base de datos deben cumplir. Una restricción importante es la **correspondencia de cardinalidades,** que expresa el número de entidades con las que otra entidad se puede asociar a través de un conjunto de relaciones. Por ejemplo, si un instructor solo se puede asociar con un único departamento, el modelo E-R puede expresar esta restricción.

El modelo entidad-relación se usa ampliamente en el diseño de bases de datos, y en el Capítulo 7 se explora en detalle.

Figura 1.3. Diagrama E-R simple.

1.6.4. Normalización

Otro método de diseño de bases de datos es usar un proceso que suele denominarse normalización. El objetivo es generar un conjunto de esquemas de relaciones que permita almacenar información sin redundancias innecesarias, pero que también permita recuperar la información con facilidad. El enfoque es diseñar esquemas que se hallen en la *forma normal* adecuada. Para determinar si un esquema de relación se halla en una de las formas normales deseadas, hace falta información adicional sobre la empresa real que se está modelando con la base de datos. El enfoque más frecuente es usar **dependencias funcionales,** que se tratan en la Sección 8.4.

Para comprender la necesidad de la normalización, obsérvese lo que puede fallar en un mal diseño de base de datos. Algunas de las propiedades no deseables de un mal diseño pueden ser:

- Repetición de la información.
- Imposibilidad de representar determinada información.

Se examinarán estos problemas con la ayuda de un diseño de bases de datos modificado para el ejemplo de nuestra universidad.

Suponga que, en lugar de tener las dos tablas, *profesor* y *departamento*, separadas, se tiene una sola tabla, *facultad*, que combina la información de las dos tablas (como puede verse en la Figura 1.4). Observe que hay dos filas de *facultad* que contienen información repetida del departamento de Historia, específicamente el edificio y el presupuesto. La repetición de la información en nuestro diseño alternativo no es deseable. Repetir la información desaprovecha espacio. Más aún, complica la actualización de la base de datos. Suponga que se desea modificar el presupuesto del departamento de Historia de 50.000 € a 46.800 €. Este cambio debe quedar reflejado en las dos filas; a diferencia del diseño original donde solo se necesita actualizar una única fila. Por tanto, las actualizaciones son más costosas en este diseño alternativo que en el original. Cuando se realiza una actualización en la base de datos alternativa, debemos asegurarnos de que se actualizan todas las filas que pertenecen al departamento de Historia. En caso contrario nuestra base de datos mostrará dos valores diferentes de presupuesto para dicho departamento.

2 El lector avispado se habrá dado cuenta que se ha eliminado el atributo *nombre_dept* del conjunto de atributos que describen la entidad *profesor*; no se trata de un error. En el Capítulo 7 se verá con detalle una explicación de por qué.

ID	*nombre*	*sueldo*	*nombre_dept*	*edificio*	*presupuesto*
22222	Einstein	95.000	Física	Watson	70.000
12121	Wu	90.000	Finanzas	Painter	120.000
32343	El Said	60.000	Historia	Painter	50.000
45565	Katz	75.000	Informática	Taylor	100.000
98345	Kim	80.000	Electrónica	Taylor	85.000
76766	Crick	72.000	Biología	Watson	90.000
10101	Srinivasan	65.000	Informática	Taylor	100.000
58583	Califieri	62.000	Historia	Painter	50.000
83821	Brandt	92.000	Informática	Taylor	100.000
15151	Mozart	40.000	Música	Packard	80.000
33456	Gold	87.000	Física	Watson	70.000
76543	Singh	80.000	Finanzas	Painter	120.000

Figura 1.4. La tabla *facultad*.

Examínese ahora el problema de la «imposibilidad de representar determinada información». Suponga que se desea crear un nuevo departamento en la universidad. En el diseño alternativo anterior no se puede representar directamente la información de un departamento (*nombre_dept*, *edificio*, *presupuesto*) a no ser que el departamento tenga al menos un profesor. Se debe a que en la tabla *facultad* se requiere que existan los valores *ID*, *nombre* y *sueldo*. Esto significa que no se puede registrar la información del nuevo departamento creado hasta haber contratado al primer profesor para el departamento.

Una solución para este problema es introducir valores **nulos.** Los valores *nulos* indican que el valor no existe (o es desconocido). Los valores desconocidos pueden ser valores *ausentes* (el valor existe, pero no se tiene la información) o valores *desconocidos* (no se sabe si el valor existe realmente o no). Como se verá más adelante, los valores nulos resultan difíciles de tratar, y es preferible no recurrir a ellos. Si no se desea tratar con valores nulos, se puede crear la información del departamento solo cuando se contrate al menos a un profesor en dicho departamento, Además, habría que eliminar esa información cuando el último profesor abandone el departamento. Claramente, esta situación no es deseable ya que, bajo el diseño original de la base de datos, la información de los departamentos debería existir haya o no profesores asociados, sin necesidad de recurrir a valores nulos.

Existe una extensa teoría de normalización que ayuda a definir formalmente qué bases de datos no son deseables y cómo obtener diseños apropiados. En el Capítulo 8 se trata el diseño de bases de datos relacionales, incluyendo el proceso de normalización.

1.7. Almacenamiento de datos y consultas

Los sistemas de bases de datos están divididos en módulos que tratan con cada una de las responsabilidades del sistema general. Los componentes funcionales de los sistemas de bases de datos pueden dividirse grosso modo en los componentes gestor de almacenamiento y procesador de consultas.

El gestor de almacenamiento es importante porque las bases de datos suelen necesitar una gran cantidad de espacio de almacenamiento. Las bases de datos corporativas tienen un tamaño que va de los centenares de gigabytes hasta, para las bases de datos de mayor tamaño, los terabytes de datos. Un gigabyte son aproximadamente 1.000 megabytes, realmente 1.024 megabytes, (1.000 millones de bytes), y un terabyte es aproximadamente un millón de megabytes (1 billón de bytes). Debido a que la memoria principal de las computadoras no puede almacenar toda esta información, la información se almacena en discos. Los datos se intercambian entre los discos de almacenamiento y la memoria principal cuando sea necesario. Como el intercambio de datos con el disco es lento, comparado con la velocidad de la unidad central de procesamiento, es fundamental que el sistema de base de datos estructure los datos para minimizar la necesidad de intercambio de datos entre los discos y la memoria principal.

El procesador de consultas es importante porque ayuda al sistema de bases de datos a simplificar y facilitar el acceso a los datos. El procesador de consultas permite que los usuarios de la base de datos consigan un buen rendimiento a la vez que pueden trabajar en el nivel de vistas sin necesidad de comprender los detalles del diseño físico de la implementación del sistema. Es función del sistema de bases de datos traducir las actualizaciones y las consultas escritas en lenguajes no procedimentales, en el nivel lógico, en una secuencia eficiente de operaciones en el nivel físico.

1.7.1. Gestor de almacenamiento

Un *gestor de almacenamiento* es un módulo de programa que proporciona la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y las consultas remitidas al sistema. El gestor de almacenamiento es responsable de la interacción con el gestor de archivos. Los datos en bruto se almacenan en el disco mediante el sistema de archivos que suele proporcionar un sistema operativo convencional. El gestor de almacenamiento traduce las diferentes instrucciones LMD a comandos de bajo nivel del sistema de archivos. Así, el gestor de almacenamiento es responsable del almacenamiento, la recuperación y la actualización de los datos de la base de datos.

Entre los componentes del gestor de almacenamiento se encuentran:

- **Gestor de autorizaciones e integridad,** que comprueba que se satisfagan las restricciones de integridad y la autorización de los usuarios para tener acceso a los datos.
- **Gestor de transacciones,** que garantiza que la base de datos quede en un estado consistente (correcto) a pesar de los fallos del sistema, y que la ejecución concurrente de transacciones transcurra sin conflictos.
- **Gestor de archivos,** que gestiona la asignación de espacio de almacenamiento de disco y las estructuras de datos usadas para representar la información almacenada en el disco.
- **Gestor de la memoria intermedia,** que es responsable de traer los datos desde el disco de almacenamiento a la memoria principal, así como de decidir los datos a guardar en la memoria caché. El gestor de la memoria intermedia es una parte fundamental de los sistemas de bases de datos, ya que permite que la base de datos maneje tamaños de datos mucho mayores que el tamaño de la memoria principal.

El gestor de almacenamiento implementa varias estructuras de datos como parte de la implementación física del sistema:

- **Archivos de datos,** que almacenan la base de datos en sí misma.
- **Diccionario de datos,** que almacena metadatos acerca de la estructura de la base de datos; en particular, su esquema.
- **Índices,** que pueden proporcionar un acceso rápido a los elementos de datos. Como el índice de este libro de texto, los índices de las bases de datos facilitan punteros a los elementos de datos que tienen un valor concreto. Por ejemplo, se puede usar un índice para buscar todos los registros *profesor* con un *ID* determinado, o todos los registros *profesor* con un *nombre* dado. La asociación (*hashing*) es una alternativa a la indexación que es más rápida en algunos casos, pero no en todos.

Se estudiarán los medios de almacenamiento, las estructuras de archivos y la gestión de la memoria intermedia en el Capítulo 10. Los métodos de acceso eficiente a los datos mediante indexación o asociación se explican en el Capítulo 11.

1.7.2. El procesador de consultas

Entre los componentes del procesador de consultas se encuentran:

- **Intérprete del LDD,** que interpreta las instrucciones del LDD y registra las definiciones en el diccionario de datos.
- **Compilador del LMD,** que traduce las instrucciones del LMD en un lenguaje de consultas a un plan de evaluación que consiste en instrucciones de bajo nivel que entienda el motor de evaluación de consultas.

  Las consultas se suelen poder traducir en varios planes de ejecución alternativos, todos los cuales proporcionan el mismo resultado. El compilador del LMD también realiza una **optimización de consultas,** es decir, elige el plan de evaluación de menor coste de entre todas las opciones posibles.
- **Motor de evaluación de consultas,** que ejecuta las instrucciones de bajo nivel generadas por el compilador del LMD.

La evaluación de las consultas se trata en el Capítulo 12, mientras que los métodos por los que el optimizador de consultas elige entre las estrategias de evaluación posibles se tratan en el Capítulo 13.

1.8. Gestión de transacciones

A menudo, varias operaciones sobre la base de datos forman una única unidad lógica de trabajo. Un ejemplo son las transferencias de fondos, como se vio en la Sección 1.2, en las que se realiza un cargo en la cuenta de un departamento (llámese *A*) y un abono en otra cuenta de otro departamento (llámese *B*). Evidentemente, resulta fundamental que, o bien tengan lugar tanto el cargo como el abono, o bien no se produzca ninguno. Es decir, la transferencia de fondos debe tener lugar por completo o no producirse en absoluto. Este requisito de todo o nada se denomina **atomicidad.** Además, resulta esencial que la ejecución de la transferencia de fondos preserve la consistencia de la base de datos. Es decir, el valor de la suma de los saldos de *A* y *B* se debe preservar. Este requisito de corrección se denomina **consistencia.** Finalmente, tras la ejecución correcta de la transferencia de fondos, los nuevos valores de las cuentas *A* y *B* deben persistir, a pesar de la posibilidad de un fallo del sistema. Este requisito de persistencia se denomina **durabilidad.**

Una **transacción** es un conjunto de operaciones que lleva a cabo una única función lógica en una aplicación de bases de datos. Cada transacción es una unidad de atomicidad y consistencia. Por tanto, se exige que las transacciones no violen ninguna restricción de consistencia de la base de datos. Es decir, si la base de datos era consistente cuando la transacción comenzó, debe ser consistente cuando la transacción termine con éxito. Sin embargo, durante la ejecución de una transacción puede ser necesario permitir inconsistencias temporalmente, ya que hay que hacer en primer lugar el cargo a *A* o el abono a *B*. Esta inconsistencia temporal, aunque necesaria, puede conducir a dificultades si se produce un fallo.

Es responsabilidad del programador definir adecuadamente las diferentes transacciones, de tal manera que cada una preserve la consistencia de la base de datos. Por ejemplo, la transacción para transferir fondos de la cuenta de un departamento *A* a la cuenta de un departamento *B* puede definirse como si estuviera compuesta de dos programas diferentes: uno que realiza el cargo en la cuenta *A* y otro que realiza el abono en la cuenta *B*. La ejecución de estos dos programas, uno después del otro, preservará realmente la consistencia. Sin embargo, cada programa en sí mismo no transforma la base de datos de un estado consistente a otro nuevo. Por tanto, estos programas no son transacciones.

Garantizar las propiedades de atomicidad y de durabilidad es responsabilidad del propio sistema de bases de datos; concretamente, del **componente de gestión de transacciones.** Si no existen fallos, todas las transacciones se completan con éxito y la atomicidad se consigue fácilmente. Sin embargo, debido a diversos tipos de fallos, puede que las transacciones no siempre completen su ejecución con éxito. Si se va a asegurar la propiedad de atomicidad, las transacciones fallidas no deben tener ningún efecto sobre el estado de la base de datos. Por tanto, esta debe restaurarse al estado en que estaba antes de que la transacción en cuestión comenzó a ejecutarse. El sistema de bases de datos, por tanto, debe realizar la **recuperación de fallos,** es decir, detectar los fallos del sistema y restaurar la base de datos al estado que tenía antes de que ocurriera el fallo.

Finalmente, cuando varias transacciones actualizan la base de datos de manera concurrente, puede que no se preserve la consistencia de los datos, aunque cada una de las transacciones sea correcta. Es responsabilidad del **gestor de control de concurrencia** controlar la interacción entre las transacciones concurrentes para garantizar la consistencia de la base de datos. El **gestor de transacciones** consta del gestor de control de concurrencia y del gestor de recuperación de fallos.

Los conceptos básicos del procesamiento de transacciones se tratan en el Capítulo 14. La gestión de las transacciones concurrentes se trata en el Capítulo 15. En el Capítulo 16 se estudiará con detalle la recuperación de fallos.

El concepto de transacción se ha manejado ampliamente en los sistemas de bases de datos y en sus aplicaciones. Aunque el empleo inicial de las transacciones se produjo en las aplicaciones financieras, el concepto se usa ahora en aplicaciones en tiempo real de telecomunicaciones, así como en la gestión de las actividades de larga duración, como el diseño de productos o los flujos de trabajo administrativo. Estas aplicaciones más amplias del concepto de transacción se estudian en el Capítulo 26.

1.9. Arquitectura de las bases de datos

Ya podemos ofrecer una visión única (Figura 1.5) de los diversos componentes de los sistemas de bases de datos y de las conexiones existentes entre ellos.

La arquitectura de los sistemas de bases de datos se ve muy influida por el sistema informático subyacente, sobre el que se ejecuta el sistema de bases de datos. Los sistemas de bases de datos pueden estar centralizados o ser del tipo cliente-servidor, en los que una máquina servidora ejecuta el trabajo en nombre de multitud de máquinas clientes. Los sistemas de bases de datos pueden dise-

ñarse también para aprovechar las arquitecturas de computadoras paralelas. Las bases de datos distribuidas se extienden por varias máquinas geográficamente separadas.

En el Capítulo 17 se trata la arquitectura general de los sistemas informáticos modernos. El Capítulo 18 describe el modo en que diversas acciones de las bases de datos, en especial el procesamiento de las consultas, pueden implementarse para aprovechar el procesamiento paralelo. El Capítulo 19 presenta varios problemas que surgen en las bases de datos distribuidas y describe el modo de afrontarlos. Entre los problemas se encuentran el modo de almacenar los datos, la manera de asegurar la atomicidad de las transacciones que se ejecutan en varios sitios, cómo llevar a cabo controles de concurrencia y el modo de ofrecer alta disponibilidad en presencia de fallos. El procesamiento distribuido de las consultas y los sistemas de directorio también se describen en ese capítulo.

Hoy en día la mayor parte de los usuarios de los sistemas de bases de datos no está presente en el lugar físico en que se encuentra el sistema de base de datos, sino que se conectan a él a través de una red. Por tanto, se puede diferenciar entre los equipos **clientes,** en los que trabajan los usuarios remotos de la base de datos, y los equipos **servidores,** en los que se ejecutan los sistemas de bases de datos.

Las aplicaciones de bases de datos suelen dividirse en dos o tres partes, como puede verse en la Figura 1.6. En una **arquitectura de dos capas,** la aplicación se divide en un componente que reside en la máquina cliente, que invoca la funcionalidad del sistema de bases de datos en la máquina servidora mediante instrucciones del lenguaje de consultas. Los estándares de interfaces de programas de aplicación, como ODBC y JDBC, se usan para la interacción entre el cliente y el servidor.

En cambio, en una **arquitectura de tres capas,** la máquina cliente actúa simplemente como una parte visible al usuario y no alberga llamadas directas a la base de datos. En vez de eso, el extremo cliente se comunica con un **servidor de aplicaciones,** generalmente mediante una interfaz de formularios. El servidor de aplicaciones, a su vez, se comunica con el sistema de bases de datos para tener acceso a los datos. La **lógica de negocio** de la aplicación, que establece las acciones que se deben realizar según las condiciones reinantes, se incorpora en el servidor de aplicaciones, en lugar de estar distribuida entre múltiples clientes. Las aplicaciones de tres capas resultan más adecuadas para aplicaciones de gran tamaño y para las aplicaciones que se ejecutan en la *World Wide Web*.

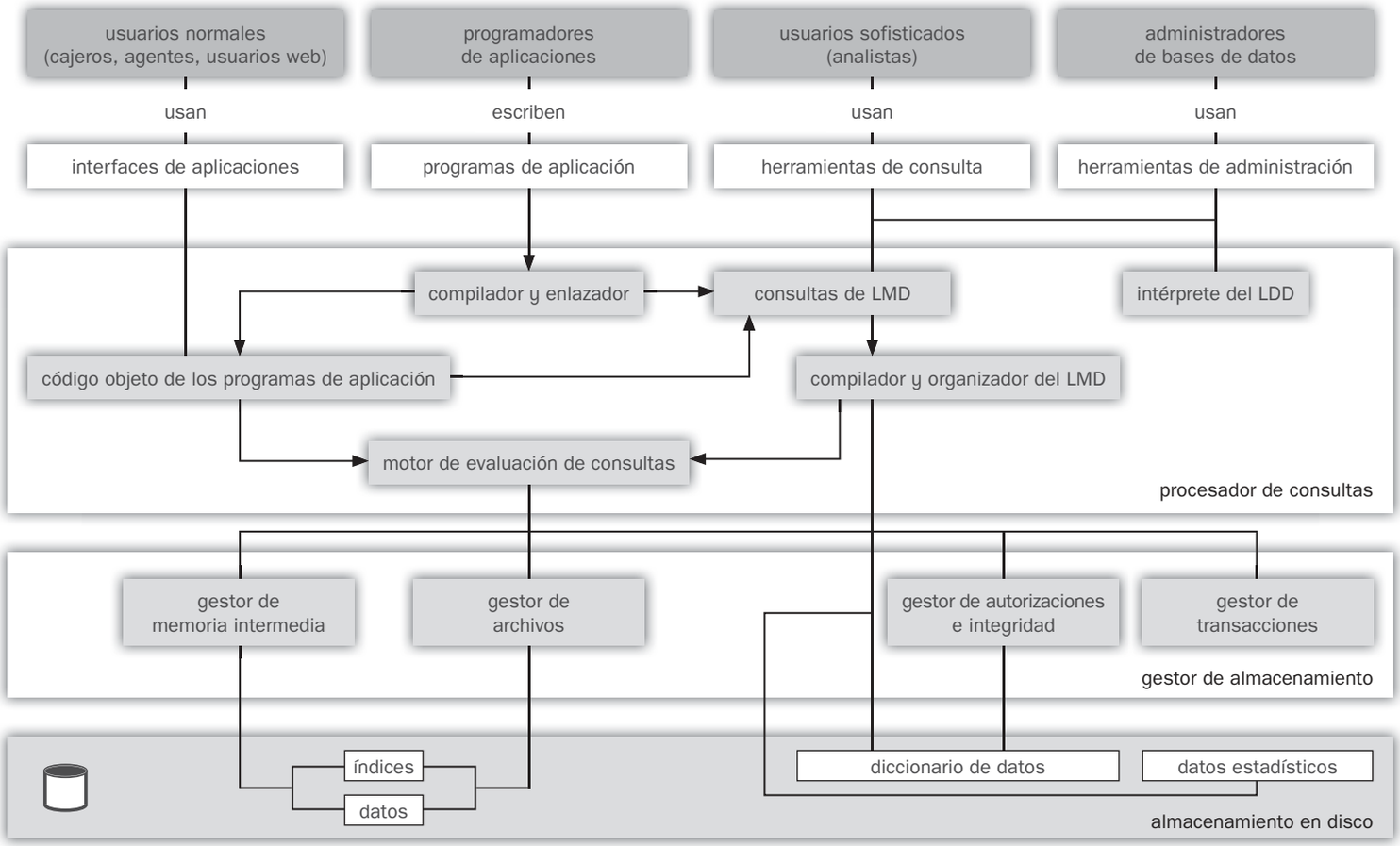
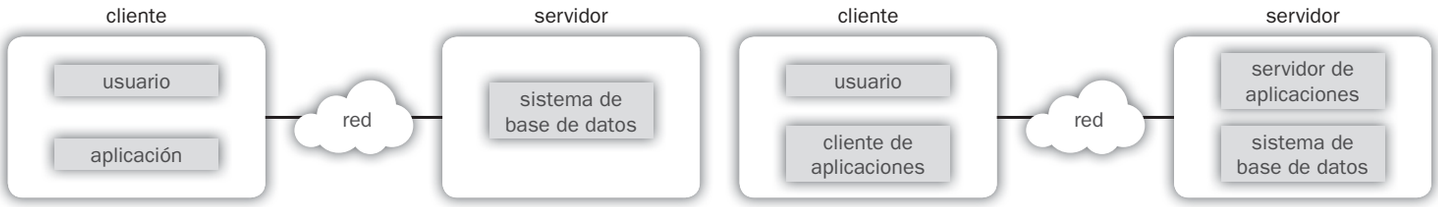


Figura 1.5. Arquitectura del sistema.



(a) Arquitectura de dos capas (b) Arquitectura de tres capas

Figura 1.6. Arquitecturas de dos y tres capas.

1.10. Minería y análisis de datos

El término **minería de datos** se refiere, en líneas generales, al proceso de análisis semiautomático de grandes bases de datos para descubrir patrones útiles. Al igual que el descubrimiento de conocimientos en la inteligencia artificial (también denominado **aprendizaje de la máquina**) o el análisis estadístico, la minería de datos intenta descubrir reglas y patrones en los datos. Sin embargo, la minería de datos se diferencia del aprendizaje de la máquina y de la estadística en que maneja grandes volúmenes de datos, almacenados principalmente en disco. Es decir, la minería de datos trata del «descubrimiento de conocimientos en las bases de datos».

Algunos tipos de conocimientos descubiertos en las bases de datos pueden representarse mediante un conjunto de **reglas.** Lo que sigue es un ejemplo de regla, definida informalmente: «las mujeres jóvenes con ingresos anuales superiores a 50.000 € son las personas con más probabilidades de comprar coches deportivos pequeños». Por supuesto, esas reglas no son universalmente ciertas, sino que tienen grados de «apoyo» y de «confianza». Otros tipos de conocimiento se representan mediante ecuaciones que relacionan diferentes variables, o mediante otros mecanismos para la predicción de los resultados cuando se conocen los valores de algunas variables.

Una gran variedad de tipos posibles de patrones pueden resultar útiles, y para descubrir tipos de patrones diferentes se emplean diversas técnicas. En el Capítulo 20 se estudian unos cuantos ejemplos de patrones y se ve la manera en que pueden obtenerse de las bases de datos de forma automática.

Generalmente, en la minería de datos hay un componente manual, que consiste en el preprocesamiento de los datos de una manera aceptable para los algoritmos, y el posprocesamiento de los patrones descubiertos para descubrir otros nuevos que puedan resultar útiles. Lo normal es que más de un tipo de patrón pueda ser descubierto en una base de datos dada, y quizá sea necesaria la interacción manual para escoger los tipos de patrones útiles. Por este motivo, la minería de datos en la vida real es, en realidad, un proceso semiautomático. No obstante, en nuestra descripción se concentra la atención en el aspecto automático de la minería.

Las empresas han comenzado a explotar la creciente cantidad de datos en línea para tomar mejores decisiones sobre sus actividades, como los artículos de los que hay que tener existencias y la mejor manera de llegar a los clientes para incrementar las ventas. Muchas de sus consultas son, sin embargo, bastante complicadas, y hay cierto tipo de información que no puede extraerse ni siquiera usando SQL.

Se dispone de varias técnicas y herramientas para ayudar a la toma de decisiones. Algunas herramientas para el análisis de datos permiten a los analistas ver los datos de diferentes maneras. Otras herramientas de análisis realizan cálculos previos de resúmenes de grandes cantidades de datos, con objeto de dar respuestas rápidas a las preguntas. El estándar SQL contiene actualmente constructores adicionales para soportar el análisis de datos.

Las grandes compañías disponen de distintas fuentes de datos que necesitan utilizar para tomar decisiones empresariales. Si desean ejecutar consultas eficientes sobre datos tan diversos, las compañías han de construir *almacenes de datos*. Los almacenes de datos recogen datos de distintas fuentes bajo un esquema unificado, en un único punto. Por tanto, proporcionan al usuario una interfaz única y uniforme con los datos.

Los datos textuales también han crecido de manera explosiva. Estos datos carecen de estructura, a diferencia de los datos rígidamente estructurados de las bases de datos relacionales. La consulta de datos textuales no estructurados se denomina *recuperación de la información*. Los sistemas de recuperación de la información tienen mucho en común con los sistemas de bases de datos; en especial, el almacenamiento y la recuperación de datos en medios de almacenamiento secundarios. Sin embargo, el énfasis en el campo de los sistemas de información es diferente del de los sistemas de bases de datos, y se concentra en aspectos como las consultas basadas en palabras clave; la relevancia de los documentos para la consulta; y el análisis, clasificación e indexación de los documentos. En los Capítulos 20 y 21 se trata la ayuda a la toma de decisiones, incluyendo el procesamiento analítico en línea, la minería de datos, los almacenes de datos y la recuperación de la información.

1.11. Bases de datos específicas

Varias áreas de aplicaciones de los sistemas de bases de datos están limitadas por las restricciones del modelo de datos relacional. En consecuencia, los investigadores han desarrollado varios modelos de datos para tratar con estos dominios de aplicación, entre otros los modelos basados en objetos y los modelos de datos semiestructurados.

1.11.1. Modelos de datos basados en objetos

El modelo de datos orientado a objetos se ha convertido en la metodología dominante de desarrollo de software. Ello ha llevado al desarrollo del **modelo de datos orientado a objetos**, que se puede ver como una ampliación del modelo E-R con las nociones de encapsulación, métodos (funciones) e identidad de objetos. La herencia, la identidad de objetos y la encapsulación (ocultamiento de información), junto con otros métodos que aportan una interfaz para con los objetos, se encuentran entre los conceptos clave de la programación orientada a objetos que han encontrado aplicación en el modelado de datos. El modelo de datos orientado a objetos también enriquece el sistema de tipos, incluyendo tipos estructurados y colecciones. En los años ochenta, se desarrollaron varios sistemas de bases de datos siguiendo un modelo de datos orientado a objetos.

Los principales proveedores de bases de datos apoyan actualmente el **modelo de datos relacional orientado a objetos** que combina las características del modelo de datos orientado a objetos con el modelo de datos relacional. Amplía el modelo relacional tradicional con distintas características como los tipos estructurados y las colecciones, así como la orientación a objetos. El Capítulo 22 trata el modelo de datos relacional orientado a objetos.

1.11.2. Modelos de datos semiestructurados

Los modelos de datos semiestructurados permiten la especificación de los datos, de modo que cada elemento de datos del mismo tipo puede tener conjuntos de atributos diferentes. Esto los diferencia de los modelos de datos mencionados anteriormente, en los que todos los elementos de datos de un tipo dado deben tener el mismo conjunto de atributos.

El lenguaje XML se diseñó inicialmente como un modo de añadir información de marcas a los documentos de texto, pero se ha vuelto importante debido a sus aplicaciones en el intercambio de datos. XML ofrece un modo de representar los datos que tienen una estructura anidada y, además, permite una gran flexibilidad en la estructuración de los datos, lo cual es importante para ciertas clases de datos no tradicionales. En el Capítulo 23 se describe el lenguaje XML, diferentes maneras de expresar las consultas sobre datos representados en XML y la transformación de los datos XML de una forma a otra.

1.12. Usuarios y administradores de bases de datos

Uno de los objetivos principales de los sistemas de bases de datos es recuperar información de la base de datos y almacenar en ella información nueva. Las personas que trabajan con una base de datos se pueden clasificar como usuarios o administradores de bases de datos.

1.12.1. Usuarios de bases de datos e interfaces de usuario

Hay cuatro tipos diferentes de usuarios de los sistemas de bases de datos, diferenciados por la forma en que esperan interactuar con el sistema. Se han diseñado diversos tipos de interfaces de usuario para los diferentes tipos de usuarios.

- Los **usuarios normales** son usuarios no sofisticados que interactúan con el sistema invocando alguno de los programas de aplicación que se han escrito previamente. Por ejemplo, un empleado de la universidad que tiene que añadir un nuevo profesor a un departamento A invoca un programa llamado *nueva_contratacion*. Este programa le pide al empleado el nombre del nuevo profesor, su nuevo *ID*, el nombre del departamento (es decir, A) y el sueldo.

  La interfaz de usuario habitual para los usuarios normales es una interfaz de formularios, en la que el usuario puede rellenar los campos correspondientes del formulario. Los usuarios normales también pueden limitarse a leer *informes* generados por la base de datos.

  Como ejemplo adicional, considérese un estudiante que durante el periodo de matrícula quiere matricularse en una asignatura utilizando una interfaz web. Ese usuario puede acceder a una aplicación web que se ejecuta en un servidor web. La aplicación comprueba en primer lugar la identidad del usuario y le permite acceder a un formulario en el que introduce la información solicitada. La información del formulario se envía de vuelta a la aplicación web en el servidor, se comprueba si hay espacio en la asignatura (realizando una consulta a la base de datos) y si lo hay, añade la información del estudiante a la asignatura en la base de datos.
- Los **programadores de aplicaciones** son profesionales informáticos que escriben programas de aplicación. Los programadores de aplicaciones pueden elegir entre muchas herramientas para desarrollar las interfaces de usuario. Las herramientas de **desarrollo rápido de aplicaciones (DRA)** son herramientas que permiten al programador de aplicaciones crear formularios e informes con un mínimo esfuerzo de programación.
- Los **usuarios sofisticados** interactúan con el sistema sin escribir programas. En su lugar, formulan sus consultas en un lenguaje de consultas de bases de datos o con herramientas como el software de análisis de datos. Los analistas que remiten las consultas para explorar los datos de la base de datos entran en esta categoría.
- Los **usuarios especializados** son usuarios sofisticados que escriben aplicaciones de bases de datos especializadas que no encajan en el marco tradicional del procesamiento de datos. Entre estas aplicaciones están los sistemas de diseño asistido por computadora, los sistemas de bases de conocimientos y los sistemas expertos, los sistemas que almacenan datos con tipos de datos complejos (por ejemplo, los datos gráficos y los datos de sonido) y los sistemas de modelado del entorno. En el Capítulo 22 se estudian varias de estas aplicaciones.

1.12.2. Administrador de bases de datos

Una de las principales razones de usar un SGBD es tener un control centralizado tanto de los datos como de los programas que tienen acceso a esos datos. La persona que tiene ese control central sobre el sistema se denomina **administrador de bases de datos (ABD).** Las funciones del ABD incluyen:

- La **definición del esquema.** El ABD crea el esquema original de la base de datos mediante la ejecución de un conjunto de instrucciones de definición de datos en el LDD.
- La **definición de la estructura y del método de acceso.**
- La **modificación del esquema y de la organización física.** El ABD realiza modificaciones en el esquema y en la organización física para reflejar las necesidades cambiantes de la organización, o para alterar la organización física a fin de mejorar el rendimiento.
- La **concesión de autorización para el acceso a los datos.** Mediante la concesión de diferentes tipos de autorización, el administrador de bases de datos puede regular las partes de la base de datos a las que puede tener acceso cada usuario. La información de autorización se guarda en una estructura especial del sistema que el SGBD consulta siempre que alguien intenta tener acceso a los datos del sistema.
- El **mantenimiento rutinario.** Algunos ejemplos de las actividades de mantenimiento rutinario de un administrador de bases de datos son:
  - Copia de seguridad periódica de la base de datos, bien sobre cinta o bien sobre servidores remotos, para impedir la pérdida de datos en casos de desastre como inundaciones.
  - Asegurarse de que se dispone de suficiente espacio libre en disco para las operaciones normales y aumentar el espacio en el disco según sea necesario.
  - Supervisar los trabajos que se ejecuten en la base de datos y asegurarse de que el rendimiento no se degrade debido a que algún usuario haya remitido tareas muy costosas.

1.13. Historia de los sistemas de bases de datos

El procesamiento de datos impulsa el crecimiento de las computadoras, como lo ha hecho desde los primeros días de las computadoras comerciales. De hecho, la automatización de las tareas de procesamiento de datos precede a las computadoras. Las tarjetas perforadas, inventadas por Herman Hollerith, se emplearon a principios del siglo XX para registrar los datos del censo de Estados Unidos, y se usaron sistemas mecánicos para procesar las tarjetas y para tabular los resultados. Las tarjetas perforadas se utilizaron posteriormente con profusión como medio para introducir datos en las computadoras.

Las técnicas de almacenamiento y de procesamiento de datos han evolucionado a lo largo de los años:

- **Años cincuenta y primeros años sesenta:** se desarrollaron las cintas magnéticas para el almacenamiento de datos. Las tareas de procesamiento de datos, como la elaboración de nóminas, se automatizaron con los datos almacenados en cintas. El procesamiento de datos consistía en leer datos de una o varias cintas y escribirlos en una nueva cinta. Los datos también se podían introducir desde paquetes de tarjetas perforadas e imprimirse en impresoras. Por ejemplo, los aumentos de sueldo se procesaban introduciéndolos en las tarjetas perforadas y leyendo el paquete de cintas perforadas de manera sincronizada con una cinta que contenía los detalles principales de los sueldos.

Los registros debían estar en el mismo orden. Los aumentos de sueldo se añadían a los sueldos leídos de la cinta maestra y se escribían en una nueva cinta; esa nueva cinta se convertía en la nueva cinta maestra.

Las cintas (y los paquetes de tarjetas perforadas) solo se podían leer secuencialmente, y el tamaño de los datos era mucho mayor que la memoria principal; por tanto, los programas de procesamiento de datos se veían obligados a procesar los datos en un orden determinado, leyendo y mezclando datos de las cintas y de los paquetes de tarjetas perforadas.

- **Finales de los años sesenta y años setenta:** el empleo generalizado de los discos duros a finales de los años sesenta modificó en gran medida la situación del procesamiento de datos, ya que permitieron el acceso directo a los datos. La ubicación de los datos en disco no era importante, ya que se podía tener acceso a cualquier posición del disco en solo unas decenas de milisegundos. Los datos se liberaron así de la tiranía de la secuencialidad. Con los discos pudieron crearse las bases de datos de red y las bases de datos jerárquicas, que permitieron que las estructuras de datos, como las listas y los árboles, pudieran almacenarse en disco. Los programadores pudieron crear y manipular estas estructuras de datos.

  El artículo histórico de Codd [1970] definió el modelo relacional y las formas no procedimentales de consultar los datos en el modelo relacional, y así nacieron las bases de datos relacionales. La simplicidad del modelo relacional y la posibilidad de ocultar completamente los detalles de implementación a los programadores resultaron realmente atractivas. Codd obtuvo posteriormente el prestigioso premio Turing de la ACM (*Association of Computing Machinery*: asociación de maquinaria informática) por su trabajo.

- **Años ochenta:** aunque académicamente interesante, el modelo relacional no se usó inicialmente en la práctica debido a sus inconvenientes en cuanto a rendimiento; las bases de datos relacionales no podían igualar el rendimiento de las bases de datos de red y jerárquicas existentes. Esta situación cambió con System R, un proyecto innovador del centro de investigación de IBM que desarrolló técnicas para la construcción de un sistema de bases de datos relacionales eficiente. En Astrahan et ál. [1976] y Chamberlin et ál. [1981] se pueden encontrar excelentes visiones generales de System R. El prototipo System R completamente funcional condujo al primer producto de bases de datos relacionales de IBM: SQL/DS. Al mismo tiempo, se estaba desarrollando el sistema Ingres en la Universidad de California, lo que condujo a un producto comercial del mismo nombre. Los primeros sistemas comerciales de bases de datos relacionales, como DB2 de IBM, Oracle, Ingres y Rdb de DEC, desempeñaron un importante papel en el desarrollo de técnicas para el procesamiento eficiente de las consultas declarativas. En los primeros años ochenta las bases de datos relacionales habían llegado a ser competitivas frente a los sistemas de bases de datos jerárquicas y de red, incluso en cuanto a rendimiento. Las bases de datos relacionales eran tan sencillas de usar que finalmente reemplazaron a las bases de datos jerárquicas y de red; los programadores que usaban esas bases de datos se veían obligados a tratar muchos detalles de implementación de bajo nivel y tenían que codificar sus consultas de forma procedimental, y lo que era aún más importante, tenían que tener presente el rendimiento durante el diseño de los programas, lo que suponía un gran esfuerzo. En cambio, en las bases de datos relacionales, casi todas estas tareas de bajo nivel las realiza de manera automática el sistema de bases de datos, lo que libera al programador para que se centre en el nivel lógico. Desde que alcanzara su liderazgo en los años ochenta, el modelo relacional ha reinado sin discusión entre todos los modelos de datos.

  Los años ochenta también fueron testigos de profunda investigación en las bases de datos paralelas y distribuidas, así como del trabajo inicial en las bases de datos orientadas a objetos.

- **Primeros años noventa:** el lenguaje SQL se diseñó fundamentalmente para las aplicaciones de ayuda a la toma de decisiones, que son intensivas en consultas, mientras que el objetivo principal de las bases de datos en los años ochenta eran las aplicaciones de procesamiento de transacciones, que son intensivas en actualizaciones. La ayuda a la toma de decisiones y las consultas volvieron a emerger como una importante área de aplicación para las bases de datos. El uso de las herramientas para analizar grandes cantidades de datos experimentó un gran crecimiento.

  En esta época muchas marcas de bases de datos introdujeron productos de bases de datos paralelas. Las diferentes marcas de bases de datos comenzaron también a añadir soporte relacional orientado a objetos a sus bases de datos.

- **Finales de los años noventa:** el principal acontecimiento fue el crecimiento explosivo de la *World Wide Web*. Las bases de datos se implantaron mucho más ampliamente que nunca hasta ese momento. Los sistemas de bases de datos tenían que soportar tasas de procesamiento de transacciones muy elevadas, así como una fiabilidad muy alta, y disponibilidad 24×7 (disponibilidad 24 horas al día y 7 días a la semana, lo que impedía momentos de inactividad debidos a actividades de mantenimiento planificadas). Los sistemas de bases de datos también tenían que soportar interfaces web para los datos.

- **Años 2000:** la primera mitad de los años 2000 ha sido testigo del crecimiento de XML y de su lenguaje de consultas asociado, XQuery, como nueva tecnología de las bases de datos. Aunque XML se emplea ampliamente para el intercambio de datos, así como para el almacenamiento de tipos de datos complejos, las bases de datos relacionales siguen siendo el núcleo de la inmensa mayoría de las aplicaciones de bases de datos muy grandes. En este periodo también se ha podido presenciar el crecimiento de las técnicas de «informática autónoma/administración automática» para la minimización del esfuerzo de administración. Este periodo también ha vivido un gran crecimiento del uso de sistemas de bases de datos de fuente abierta, en particular PostgreSQL y MySQL.

  En la última parte de la década se ha visto un crecimiento de las bases de datos especializadas en el análisis de datos, en particular el almacén de columnas, que guardan realmente cada columna de una tabla como *array* (matriz) separado, y sistemas de bases de datos masivamente paralelos diseñados para el análisis de grandes conjuntos de datos. Se han construido varios sistemas de almacenamiento distribuido novedosos para el manejo de los requisitos de gestión de datos de sitios web de grandes dimensiones, como Amazon, Facebook, Google, Microsoft y Yahoo!, y algunos de ellos se ofertan como servicios web que pueden utilizar desarrolladores de aplicaciones. También se ha realizado un importante trabajo en la gestión y análisis de flujos de datos, como los datos del mercado de valores o los datos de gestión de las redes de computadores. Se han desarrollado ampliamente técnicas de minería de datos; entre las aplicaciones de ejemplo se encuentran los sistemas de recomendación de productos para la web y la selección automática de anuncios relevantes en páginas web.

1.14. Resumen

- Un **sistema gestor de bases de datos (SGBD)** consiste en un conjunto de datos interrelacionados y un conjunto de programas para tener acceso a esos datos. Los datos describen una empresa concreta.
- El objetivo principal de un SGBD es proporcionar un entorno que sea tanto conveniente como eficiente para las personas que lo usan, para la recuperación y almacenamiento de información.
- Los sistemas de bases de datos resultan ubicuos hoy en día, y la mayor parte de la gente interactúa, directa o indirectamente, con bases de datos muchas veces al día.
- Los sistemas de bases de datos se diseñan para almacenar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para el almacenamiento de la información como la provisión de mecanismos para la manipulación de la información. Además, los sistemas de bases de datos deben preocuparse de la seguridad de la información almacenada, en caso de caídas del sistema o de intentos de acceso sin autorización. Si los datos deben compartirse entre varios usuarios, el sistema debe evitar posibles resultados anómalos.
- Uno de los propósitos principales de los sistemas de bases de datos es ofrecer a los usuarios una visión abstracta de los datos. Es decir, el sistema oculta ciertos detalles de la manera en que los datos se almacenan y mantienen.
- Por debajo de la estructura de la base de datos se halla el **modelo de datos:** un conjunto de herramientas conceptuales para describir los datos, las relaciones entre ellos, la semántica de los datos y las restricciones de estos.
- El modelo de datos relacional es el modelo más atendido para el almacenamiento de datos en bases de datos. Otros modelos son el modelo orientado a objetos, el modelo relacional orientado a objetos y los modelos de datos semiestructurados.
- Un **lenguaje de manipulación de datos (LMD)** es un lenguaje que permite a los usuarios tener acceso a los datos o manipularlos. Los LMD no procedimentales, que solo necesitan que el usuario especifique los datos que necesita, sin aclarar exactamente la manera de obtenerlos, se usan mucho hoy en día.
- Un **lenguaje de definición de datos (LDD)** es un lenguaje para la especificación del esquema de la base de datos y otras propiedades de los datos.
- El diseño de bases de datos supone sobre todo el diseño del esquema de la base de datos. El modelo de datos entidad-relación (E-R) es un modelo de datos muy usado para el diseño de bases de datos. Proporciona una representación gráfica conveniente para ver los datos, las relaciones y las restricciones.
- Un sistema de bases de datos consta de varios subsistemas:
  - El subsistema **gestor de almacenamiento** proporciona la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación, así como las consultas remitidas al sistema.
  - El subsistema **procesador de consultas** compila y ejecuta instrucciones LDD y LMD.
- El **gestor de transacciones** garantiza que la base de datos permanezca en un estado consistente (correcto) a pesar de los fallos del sistema. El gestor de transacciones garantiza que la ejecución de las transacciones concurrentes se produzca sin conflictos.
- La arquitectura de un sistemas de base de datos se ve muy influida por el sistema computacional subyacente en que se ejecuta el sistema de la base de datos. Pueden ser centralizados o cliente-servidor, en el que una computadora servidora ejecuta el trabajo para múltiples computadoras cliente. Los sistemas de bases de datos también se diseñan para explotar arquitecturas computacionales paralelas. Las bases de datos distribuidas pueden extenderse geográficamente en múltiples computadoras separadas.
- Las aplicaciones de bases de datos suelen dividirse en un *front-end*, que se ejecuta en las máquinas clientes, y una parte que se ejecuta en *back-end*. En las arquitecturas de dos capas, el *front-end* se comunica directamente con una base de datos que se ejecuta en el *back-end*. En las arquitecturas de tres capas, la parte en *back-end* se divide a su vez en un servidor de aplicaciones y un servidor de bases de datos.
- Las técnicas de descubrimiento de conocimientos intentan descubrir automáticamente reglas y patrones estadísticos a partir de los datos. El campo de la minería de datos combina las técnicas de descubrimiento de conocimientos desarrolladas por los investigadores de inteligencia artificial y analistas estadísticos con las técnicas de implementación eficiente que les permiten utilizar bases de datos extremadamente grandes.
- Existen cuatro tipos diferentes de usuarios de sistemas de bases de datos, diferenciándose por las expectativas de interacción con el sistema. Se han desarrollado diversos tipos de interfaces de usuario diseñadas para los distintos tipos de usuarios.

Términos de repaso

- Sistema gestor de bases de datos (SGBD).
- Aplicaciones de sistemas de bases de datos.
- Sistemas procesadores de archivos.
- Inconsistencia de datos.
- Restricciones de consistencia.
- Abstracción de datos.
- Ejemplar de la base de datos.
- Esquema.
  - Esquema físico.
  - Esquema lógico.
- Independencia física de los datos.
- Modelos de datos.
  - Modelo entidad-relación.
  - Modelo de datos relacional.
  - Modelo de datos basado en objetos.
  - Modelo de datos semiestructurados.
- Lenguajes de bases de datos.
  - Lenguaje de definición de datos.
  - Lenguaje de manipulación de datos.
  - Lenguaje de consultas.
- Metadatos.
- Programa de aplicación.
- Normalización.
- Diccionario de datos.
- Gestor de almacenamiento.
- Procesador de consultas.
- Transacciones.
  - Atomicidad.
  - Recuperación de fallos.
  - Control de concurrencia.
- Arquitecturas de bases de datos de dos y tres capas.
- Minería de datos.
- Administrador de bases de datos (ABD).

Ejercicios prácticos

1.1. En este capítulo se han descrito varias ventajas importantes de los sistemas gestores de bases de datos. ¿Cuáles son sus dos inconvenientes?

1.2. Indique cinco formas en que los sistemas de declaración de tipos de lenguajes, como Java o C++, difieren de los lenguajes de definición de datos utilizados en una base de datos.

1.3. Indique seis pasos importantes que se deben dar para configurar una base de datos para una empresa dada.

1.4. Indique al menos tres tipos diferentes de información que una universidad debería mantener, además de la indicada en la Sección 1.6.2.

1.5. Suponga que quiere construir un sitio web similar a YouTube. Tenga en cuenta todos los puntos indicados en la Sección 1.2, como las desventajas de mantener los datos en un sistema de procesamiento de archivos. Justifique la importancia de cada uno de esos puntos para el almacenamiento de datos de vídeo real y de los metadatos del vídeo, como su título, el usuario que lo subió y los usuarios que lo han visto.

1.6. Las consultas por palabras clave que se usan en las búsquedas web son muy diferentes de las consultas de bases de datos. Indique las diferencias más importantes entre las dos, en términos de las formas en que se especifican las consultas y cómo son los resultados de una consulta.

Ejercicios

1.7. Indique cuatro aplicaciones que se hayan usado y que sea muy posible que utilicen un sistema de bases de datos para almacenar datos persistentes.

1.8. Indique cuatro diferencias significativas entre un sistema de procesamiento de archivos y un SGBD.

1.9. Explique el concepto de independencia física de los datos y su importancia en los sistemas de bases de datos.

1.10. Indique cinco responsabilidades del sistema gestor de bases de datos. Para cada responsabilidad, explique los problemas que surgirían si no se asumiera esa responsabilidad.

1.11. Indique al menos dos razones para que los sistemas de bases de datos soporten la manipulación de datos mediante un lenguaje de consultas declarativo, como SQL, en vez de limitarse a ofrecer una biblioteca de funciones de C o de C++ para llevar a cabo la manipulación de los datos.

1.12. Explique los problemas que causa el diseño de la tabla de la Figura 1.4.

1.13. ¿Cuáles son las cinco funciones principales del administrador de bases de datos?

1.14. Explique la diferencia entre la arquitectura en dos capas y en tres capas. ¿Cuál se acomoda mejor a las aplicaciones web? ¿Por qué?

1.15. Describa al menos tres tablas que se puedan utilizar para almacenar información en un sistema de redes sociales como Facebook.

Herramientas

Hay gran número de sistemas de bases de datos comerciales actualmente en uso. Entre los principales están: DB2 de IBM (www.ibm.com/software/data/db2), Oracle (www.oracle.com), SQL Server de Microsoft (www.microsoft.com/SQL), Sybase (www.sybase.com) e IBM Informix (www.ibm.com/software/data/informix). Algunos de estos sistemas están disponibles gratuitamente para uso personal o no comercial, o para desarrollo, pero no para su implantación real.

También hay una serie de sistemas de bases de datos gratuitos o de dominio público; los más usados son MySQL (www.mySQL.com) y PostgreSQL (www.postgreSQL.org).

Una lista más completa de enlaces a sitios web de fabricantes y a otras informaciones se encuentra disponible en la página inicial de este libro, en www.db-book.com.

Notas bibliográficas

A continuación se ofrece una relación de libros de propósito general, colecciones de artículos de investigación y sitios web sobre bases de datos. Los capítulos siguientes ofrecen referencias a materiales sobre cada tema descrito en ese capítulo.

Codd [1970] es el artículo histórico que introdujo el modelo relacional.

Entre los libros de texto que tratan los sistemas de bases de datos están: Abiteboul et ál. [1995], O'Neil y O'Neil [2000], Ramakrishnan y Gehrke [2002], Date [2003], Kifer et ál. [2005], Elmasri y Navathe [2006] y García-Molina et ál. [2008]. Un libro con artículos de investigación sobre gestión de base de datos se puede encontrar en Hellerstein and Stonebraker [2005].

Un repaso de los logros en la gestión de bases de datos y una valoración de los desafíos en la investigación futura aparecen en Silberschatz et ál. [1990], Silberschatz et ál. [1996], Bernstein et ál. [1998], Abiteboul et ál. [2003] y Agrawal et ál. [2009]. La página inicial del grupo de interés especial de la ACM en gestión de datos (www.acm.org/sigmod) ofrece gran cantidad de información sobre la investigación en bases de datos. Los sitios web de los fabricantes de bases de datos (véase *Herramientas* a la izquierda de estas *Notas*) proporciona detalles acerca de sus respectivos productos.

Parte 1
Bases de datos relacionales

Un modelo de datos es un conjunto de herramientas conceptuales para la descripción de los datos, las relaciones entre ellos, su semántica y las restricciones de consistencia. Esta parte centra la atención en el modelo relacional.

El modelo de datos relacional, que se trata en el Capítulo 2, utiliza una colección de tablas para representar tanto los datos como las relaciones entre ellos. Su simplicidad conceptual le ha permitido una amplia adopción; en la actualidad la inmensa mayoría de los productos de bases de datos utilizan el modelo relacional. El modelo relacional describe los datos en los niveles lógico y de vista, abstrayendo los detalles de bajo nivel sobre el almacenamiento de los datos. El modelo entidad relación que se trata en el Capítulo 7 (Parte 2), es un modelo de datos de alto nivel que se utiliza ampliamente para el diseño de bases de datos.

Para poner a disposición de los usuarios los datos de una base de datos relacional hay que abordar varios aspectos. El aspecto más importante es determinar cómo los usuarios pueden recuperar y actualizar los datos; se han desarrollado varios lenguajes de consulta para ello. Un segundo aspecto, pero también importante, trata sobre la integridad y la protección; hay que proteger las bases de datos del posible daño por acciones del usuario, sean intencionadas o no.

Los Capítulos 3, 4 y 5 tratan sobre el lenguaje SQL, que es el lenguaje de consultas más utilizado hoy en día. En los Capítulos 3 y 4 también se realiza una descripción introductoria e intermedia de SQL. El Capítulo 4 trata así mismo de las restricciones de integridad que fuerza la base de datos y de los mecanismos de autorización, que controlan el acceso y las acciones de actualización que puede llevar a cabo un usuario. El Capítulo 5 trata temas más avanzados, incluyendo el acceso a SQL desde lenguajes de programación y el uso de SQL para el análisis de datos.

El Capítulo 6 trata sobre tres lenguajes de consulta formales: el álgebra relacional, el cálculo relacional de tuplas y el cálculo relacional de dominios, que son leguajes de consulta declarativos basados en lógica matemática. Estos lenguajes formales son la base de SQL y de otros lenguajes sencillos para el usuario, QBI y Datalog, que se describen en el Apéndice B (disponible en línea en db-book.com).

Introducción al modelo relacional

02

El modelo relacional es hoy en día el principal modelo de datos para las aplicaciones comerciales de procesamiento de datos. Ha conseguido esa posición destacada debido a su simplicidad, lo cual facilita el trabajo del programador en comparación con los modelos anteriores, como el de red y el jerárquico.

En este capítulo se estudian en primer lugar los fundamentos del modelo relacional. Existe una amplia base teórica para las bases de datos relacionales. En el Capítulo 6 se estudia la parte de esa base teórica referida a las consultas. En los Capítulos 7 y 8 se examinarán aspectos de la teoría de las bases de datos relacionales que ayudan en el diseño de esquemas de bases de datos relacionales, mientras que en los Capítulos 12 y 13 se estudian aspectos de la teoría que se refieren al procesamiento eficiente de consultas.

2.1. La estructura de las bases de datos relacionales

Una base de datos relacional consiste en un conjunto de **tablas,** a cada una de las cuales se le asigna un nombre único. Por ejemplo, suponga la tabla *profesor* de la Figura 2.1, que guarda información de todos los profesores. La tabla tiene cuatro columnas, *ID*, *nombre*, *nombre_dept* y *sueldo*. Cada fila de la tabla registra información de un profesor que consiste en el ID del profesor, su nombre, su nombre de departamento y su sueldo. De forma análoga, la tabla *asignatura*, de la Figura 2.2, guarda información de las asignaturas, que consisten en los valores *asignatura_id*, *nombre*, *nombre_dept* y *créditos* de cada asignatura. Fíjese que a cada profesor se le identifica por el valor de la columna *ID*, mientras que cada asignatura se identifica con el valor de su columna *asignatura_id*.

ID	*nombre*	*nombre_dept*	*sueldo*
10101	Srinivasan	Informática	65000
12121	Wu	Finanzas	90000
15151	Mozart	Música	40000
22222	Einstein	Física	95000
32343	El Said	Historia	60000
33456	Gold	Física	87000
45565	Katz	Informática	75000
58583	Califieri	Historia	62000
76543	Singh	Finanzas	80000
76766	Crick	Biología	72000
83821	Brandt	Informática	92000
98345	Kim	Electrónica	80000

Figura 2.1. La relación *profesor*.

La Figura 2.3 muestra una tercera tabla, *prerreq*, que guarda las asignaturas prerrequisito de otra asignatura. La tabla tiene dos columnas, *asignatura_id* y *prerreq_id*. Cada fila consta de un par de identificadores de asignatura, de forma que la segunda es un prerrequisito para la primera.

Por tanto, en una fila de la tabla *prerreq* se indica que dos asignaturas están *relacionadas* en el sentido de que una de ellas es un prerrequisito para la otra. Como otro ejemplo, considere la tabla *profesor*, en la que se puede pensar que una fila de la tabla representa la relación entre un *ID* específico y los correspondientes valores para el *nombre*, el nombre de departamento (*nombre_dept*) y el *sueldo*.

asignatura_id	*nombre*	*nombre_dept*	*créditos*
BIO-101	Introducción a la Biología	Biología	4
BIO-301	Genética	Biología	4
BIO-399	Biología computacional	Biología	3
CS-101	Introducción a la Informática	Informática	4
CS-190	Diseño de juegos	Informática	4
CS-315	Robótica	Informática	3
CS-319	Procesado de imágenes	Informática	3
CS-347	Fundamentos de bases de datos	Informática	3
EE-181	Intro. a los sistemas digitales	Electrónica	3
FIN-201	Banca de inversión	Finanzas	3
HIS-351	Historia mundial	Historia	3
MU-199	Producción de música y vídeo	Música	3
PHY-101	Fundamentos de Física	Física	4

Figura 2.2. La relación *asignatura*.

asignatura_id	*prerreq_id*
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Figura 2.3. La relación *prerreq*.

En general, una fila de una tabla representa una *relación* entre un conjunto de valores. Como una tabla es una colección de estas relaciones, existe una fuerte correspondencia entre el concepto de *tabla* y el concepto matemático de *relación*, de donde toma su nombre el modelo de datos relacional. En terminología matemática, una *tupla* es sencillamente una secuencia (o lista) de valores. Una relación entre *n* valores se representa matemáticamente como una ***n-tupla*** de valores, es decir, como una tupla con *n* valores, que se corresponde con una fila de una tabla.

En el modelo relacional, el término **relación** se utiliza para referirse a una tabla, mientras el término **tupla** se utiliza para referirse a una fila. De forma similar, el término **atributo** se refiere a una columna de una tabla.

En la Figura 2.1 se puede ver que la relación *profesor* tiene cuatro atributos: *ID*, *nombre*, *nombre_dept* y *sueldo*.

Se utiliza el término **ejemplar de relación** para referirse a una instancia específica de una relación, es decir, que contiene un determinado conjunto de filas. La instancia de *profesor* que se muestra en la Figura 2.1 tiene 12 tuplas, correspondientes a 12 profesores.

En este capítulo se verán distintas relaciones para tratar los distintos conceptos sobre el modelo de datos relacional. Estas relaciones representan parte de una universidad. No se trata de toda la información que tendría una base de datos de una universidad real, para simplificar la representación. En los Capítulos 7 y 8 se tratarán con mayor detalle los criterios para definir estructuras relacionales.

El orden en que las tuplas aparecen en una relación es irrelevante, ya que la relación es un *conjunto* de tuplas. Por tanto, si las tuplas de una relación se listan en un determinado orden, como en la Figura 2.1, o están desordenadas, como en la Figura 2.4, no tiene ninguna importancia; la relación de las dos figuras es la misma, ya que ambas contienen las mismas tuplas. Para facilitar la presentación, normalmente se mostrarán las relaciones ordenadas por su primer atributo.

ID	*nombre*	*nombre_dept*	*sueldo*
22222	Einstein	Física	95000
12121	Wu	Finanzas	90000
32343	El Said	Historia	60000
45565	Katz	Informática	75000
98345	Kim	Electrónica	80000
76766	Crick	Biología	72000
10101	Srinivasan	Informática	65000
58583	Califieri	Historia	62000
83821	Brandt	Informática	92000
15151	Mozart	Música	40000
33456	Goid	Física	87000
76543	Singh	Finanzas	80000

Figura 2.4. Lista no ordenada de la relación *profesor*.

Para cada atributo de una relación existe un conjunto de valores permitidos, llamado **dominio** del atributo. Por tanto, el dominio del atributo *sueldo* de la relación *profesor* es el conjunto de todos los posibles valores de sueldo, mientras que el dominio del atributo *nombre* es el conjunto de todos los posibles nombres de profesor.

Es necesario que, para toda relación *r*, los dominios de todos los atributos de *r* sean atómicos. Un dominio es **atómico** si los elementos del dominio se consideran unidades indivisibles. Por ejemplo, suponga que la tabla *profesor* tiene un atributo *número_teléfono*, que puede almacenar un conjunto de números de teléfono correspondientes al profesor. Entonces, el dominio de *número_teléfono* no sería atómico, ya que un elemento del dominio es un conjunto de números de teléfono, y tiene subpartes, que son cada uno de los números de teléfono individuales del conjunto.

Lo importante no es cuál es el dominio sino cómo utilizar los elementos del dominio en la base de datos. Suponga que el atributo *número_teléfono* almacena un único número de teléfono. Incluso en este caso, si se divide el valor del teléfono en un código de país, un código de área y un número local, se trataría como un valor no atómico. Si se trata cada número de teléfono como un valor único e indivisible, entonces el atributo *número_teléfono* sería un dominio atómico.

En este capítulo, así como en los Capítulos 3 al 6, se supone que todos los atributos tienen dominios atómicos. En el Capítulo 22 se tratarán las extensiones al modelo de datos relacional para permitir dominios no atómicos.

El valor ***null*** **(nulo)** es un valor especial que significa que el valor es desconocido o no existe. Por ejemplo, suponga como antes que se incluye el atributo *número_teléfono* en la relación *profesor*. Puede que un profesor no tenga número de teléfono o que no se conozca. Entonces se debería utilizar el valor *null* para indicar que el valor es desconocido o no existe. Más adelante se verá que el valor *null* genera distintas dificultades cuando se accede o actualiza la base de datos y, por tanto, si fuese posible se debería eliminar. Se supondrá inicialmente que no existen valores *null*, y en la Sección 3.6 se describe el efecto de los valores nulos sobre las distintas operaciones.

2.2. Esquema de la base de datos

Cuando se habla de bases de datos se debe diferenciar entre el **esquema de la base de datos,** que es el diseño lógico de la misma, y el **ejemplar de la base de datos,** que es una instantánea de los datos de la misma en un momento dado.

El concepto de relación se corresponde con el concepto de variable de los lenguajes de programación. El concepto **esquema de la relación** se corresponde con el concepto de definición de tipos de los lenguajes de programación.

En general, los esquemas de las relaciones consisten en una lista de los atributos y de sus dominios correspondientes. La definición exacta del dominio de cada atributo no será relevante hasta que se estudie el lenguaje SQL en el Capítulo 3.

El concepto de ejemplar de la relación se corresponde con el concepto de valor de una variable en los lenguajes de programación. El valor de una variable dada puede cambiar con el tiempo; de manera parecida, el contenido del ejemplar de una relación puede cambiar con el tiempo cuando la relación se actualiza. Sin embargo, el esquema de una relación normalmente no cambia.

Aunque es importante conocer la diferencia entre un esquema de relación y un ejemplar de relación, normalmente se utiliza el mismo nombre, como por ejemplo *profesor*, para referirse tanto a uno como a otro. Cuando es necesario, explícitamente se indica esquema o ejemplar, por ejemplo «el esquema *profesor*», o «un ejemplar de la relación *profesor*». Sin embargo, donde quede claro si significa esquema o ejemplar, simplemente se utiliza el nombre de la relación.

Suponga la relación *departamento* de la Figura 2.5. El esquema de dicha relación es:

departamento (*nombre_dept*, *edificio*, *presupuesto*)

nombre_dept	*edificio*	*sueldo*
Biología	Watson	90000
Informática	Taylor	100000
Electrónica	Taylor	85000
Finanzas	Painter	120000
Historia	Painter	50000
Música	Packard	80000
Física	Watson	70000

Figura 2.5. La relación *departamento*.

Tenga en cuenta que el atributo *nombre_dept* aparece tanto en el esquema *profesor* como en el esquema *departamento*. Esta duplicación no es una coincidencia, sino que utilizar atributos comunes en esquemas de relación es una forma de relacionar tuplas de relaciones diferentes. Por ejemplo, suponga que se desea encontrar información sobre todos los profesores que trabajan en el edificio Watson. En primer lugar se busca en la relación *departamento* los *nombre_dept* de todos los departamentos que se encuentran en el edificio Watson. A continuación, para cada relación *departamento*, se busca en la relación *profesor* para ver la información de los profesores asociados con el correspondiente *nombre_dept*.

Sigamos con el ejemplo de base de datos de una universidad.

Las asignaturas de una universidad se pueden ofertar en diversos momentos, durante distintos semestres, o incluso en un único semestre. Se necesita una relación que describa cada una de estas ofertas, o sección, de una asignatura. El esquema es:

sección (*asignatura_id*, *secc_id*, *semestre*, *año*, *edificio*, *aula*, *franja_horaria*)

La Figura 2.6 muestra un ejemplar de la relación *sección*.

Se necesita una relación que describa la asociación entre profesor y las secciones de asignaturas que imparte. El esquema de relación que describe esta asociación es:

enseña (ID, *asignatura_id*, *secc_id*, *semestre*, *año*)

La Figura 2.7 muestra un ejemplar de la relación *enseña*.

Como cabe imaginar, existen muchas más relaciones en una base de datos real de una universidad. Además de estas relaciones que ya se han visto, *profesor*, *departamento*, *asignatura*, *sección*, *prerreq* y *enseña*, se utilizarán las siguientes relaciones en el texto:

- *estudiante (ID*, *nombre*, *nombre_dept*, *total_créditos)*
- *tutor (e_id*, *p_id)*
- *matricula (ID*, *asignatura_id*, *secc_id*, *semestre*, *año*, *nota)*
- *aula (edificio*, *número_aula*, *capacidad)*
- *franja_horaria (franja_horaria_id*, *día*, *hora_inicio*, *hora_fin)*

asignatura_id	secc_id	semestre	año	edificio	aula	franja_horaria
BIO-101	1	Verano	2009	Painter	514	B
BIO-301	1	Verano	2010	Painter	514	A
CS-101	1	Otoño	2009	Packard	101	H
CS-101	1	Primavera	2010	Packard	101	F
CS-190	1	Primavera	2009	Taylor	3128	E
CS-190	2	Primavera	2009	Taylor	3128	A
CS-315	1	Primavera	2010	Watson	120	D
CS-319	1	Primavera	2010	Watson	100	B
CS-319	2	Primavera	2010	Taylor	3128	C
CS-347	1	Otoño	2009	Taylor	3128	A
EE-181	1	Primavera	2009	Taylor	3128	C
FIN-201	1	Primavera	2010	Packard	101	B
HIS-351	1	Primavera	2010	Painter	514	C
MU-199	1	Primavera	2010	Packard	101	D
PHY-101	1	Otoño	2009	Watson	100	A

Figura 2.6. La relación *sección*.

ID	asignatura_id	secc_id	semestre	año
10101	CS-101	1	Otoño	2009
10101	CS-315	1	Primavera	2010
10101	CS-347	1	Otoño	2009
12121	FIN-201	1	Primavera	2010
15151	MU-199	1	Primavera	2010
22222	PHY-101	1	Otoño	2009
32343	HIS-351	1	Primavera	2010
45565	CS-101	1	Primavera	2010
45565	CS-319	1	Primavera	2010
76766	BIO-101	1	Verano	2009
76766	BIO-301	1	Verano	2010
83821	CS-190	1	Primavera	2009
83821	CS-190	2	Primavera	2009
83821	CS-319	2	Primavera	2010
98345	EE-181	1	Primavera	2009

Figura 2.7. La relación *enseña*.

2.3. Claves

Es necesario disponer de un modo de especificar la manera en que las tuplas de una relación dada se distingan entre sí. Esto se expresa en términos de sus atributos. Es decir, los valores de los atributos de una tupla deben ser tales que puedan *identificarla unívocamente*. En otras palabras, no se permite que dos tuplas de una misma relación tengan exactamente los mismos valores en todos sus atributos.

Una **superclave** es un conjunto de uno o varios atributos que, considerados conjuntamente, permiten identificar de manera unívoca una tupla de la relación. Por ejemplo, el atributo *ID* de la relación *profesor* es suficiente para distinguir una tupla instructor de otra. Por tanto, *ID* es una superclave. El atributo *nombre* de un *profesor*, por otra parte, no es una superclave porque podría haber varios profesores con el mismo nombre.

Formalmente, sea R un conjunto de atributos en un esquema de relación r. Si se dice que un subconjunto K de R es una *superclave* de r, se restringe a ejemplares de la relación r en las que no existen dos tuplas distintas con los mismos valores en todos los atributos de K. Es decir, si existen t_1 y t_2 en r y $t_1 \neq t_2$, entonces $t_1.K \neq t_2.K$.

Una superclave puede contener atributos externos. Por ejemplo, la combinación de *ID* y *nombre* es una superclave para la relación *profesor*. Si K es una superclave, también lo es cualquier superconjunto de K. Normalmente nos interesan las superclaves para las que no hay subconjuntos que sean superclaves. Estas superclaves mínimas se denominan **claves candidatas.**

Es posible que varios conjuntos diferentes de atributos puedan ejercer como claves candidatas. Suponga que una combinación de *nombre* y *nombre_dept* es suficiente para distinguir entre los miembros de la relación *profesor*. Entonces, tanto {*ID*} como {*nombre*, *nombre_dept*} son claves candidatas. Aunque los atributos *ID* y *nombre* juntos pueden distinguir las distintas tuplas *profesor*, su combinación {*ID*, *nombre*} no forma una clave candidata, ya que el atributo *ID* por sí solo ya lo es.

Se usará el término **clave primaria** para denotar una clave candidata que ha elegido el diseñador de la base de datos como medio principal para la identificación de las tuplas de una relación. Las claves (sean primarias, candidatas o superclaves) son propiedades de toda la relación, no de cada una de las tuplas. Ninguna pareja de tuplas de la relación puede tener simultáneamente el mismo valor de los atributos de la clave. La selección de una clave representa una restricción de la empresa del mundo real que se está modelando.

Las claves candidatas deben escogerse con cuidado. Como se ha indicado, el nombre de una persona evidentemente no es suficiente, ya que puede haber muchas personas con el mismo nombre. En Estados Unidos, el atributo número de la Seguridad Social de cada persona sería una clave candidata. Dado que los residentes extranjeros no suelen tener número de la seguridad social, las empresas internacionales deben generar sus propios identificadores unívocos. Una alternativa es usar como clave alguna combinación exclusiva de otros atributos.

La clave primaria debe escogerse de manera que los valores de sus atributos no se modifiquen nunca, o muy rara vez. Por ejemplo, el campo domicilio de una persona no debe formar parte de la clave primaria, ya que es probable que se modifique. Por otra parte, está garantizado que los números de la Seguridad Social no cambian nunca. Los identificadores exclusivos generados por las empresas no suelen cambiar, salvo si se produce una fusión entre dos de ellas; en ese caso, puede que el mismo identificador haya sido emitido por ambas empresas, y puede ser necesaria una reasignación de identificadores para garantizar que sean únicos.

Los atributos de clave primaria de un esquema de relación se indican antes que el resto de los atributos; por ejemplo, el atributo *nombre_dept* del departamento aparece el primero, ya que es la clave primaria. Los atributos de la clave primaria se indican subrayados.

El esquema de una relación, por ejemplo r_1, puede incluir entre sus atributos la clave primaria de otro esquema de relación, por ejemplo r_2. Este atributo se denomina **clave externa** de r_1, que hace referencia a r_2. La relación r_1 también se denomina **relación referenciante** de la dependencia de clave externa, y r_2 se denomina **relación referenciada** de la clave externa. Por ejemplo, el atributo *nombre_dept* de *profesor* es una clave externa de *profesor* que hace referencia a *departamento*, ya que *nombre_dept* es la clave primaria de *departamento*. En cualquier ejemplar de la base de datos, dada cualquier tupla, por ejemplo t_a, de la relación *profesor*, debe haber alguna tupla, por ejemplo t_b, en la relación *departamento* tal que el valor del atributo *nombre_dept* de t_a sea el mismo que el valor de la clave primaria de t_b *nombre_dept*.

Suponga ahora las relaciones *sección* y *enseña*. Sería razonable exigir que si existe una sección para una asignatura, deba enseñarla al menos un profesor; sin embargo, podría ser que la enseñase más de un profesor. Para forzar esta restricción, se debería requerir que si una determinada combinación (*asignatura_id*, *secc_id*, *semestre*, *año*) aparece en una sección, entonces esa misma combinación aparezca en la relación *enseña*. Sin embargo, este conjunto de valores no es una clave primaria de *enseña*, ya que puede haber más de un *profesor* que enseñe dicha *sección*. Por tanto, no se puede declarar una restricción de clave externa de *sección* a *enseña* (aunque se puede definir una restricción de clave externa en la otra dirección, de *enseña* a *sección*).

La restricción de *sección* a *enseña* es un ejemplo de **restricción de integridad referencial;** una restricción de integridad referencial requiere que los valores que aparecen en determinados atributos de una tupla en la relación referenciante también aparezcan en otros atributos de al menos una tupla de la relación referenciada.

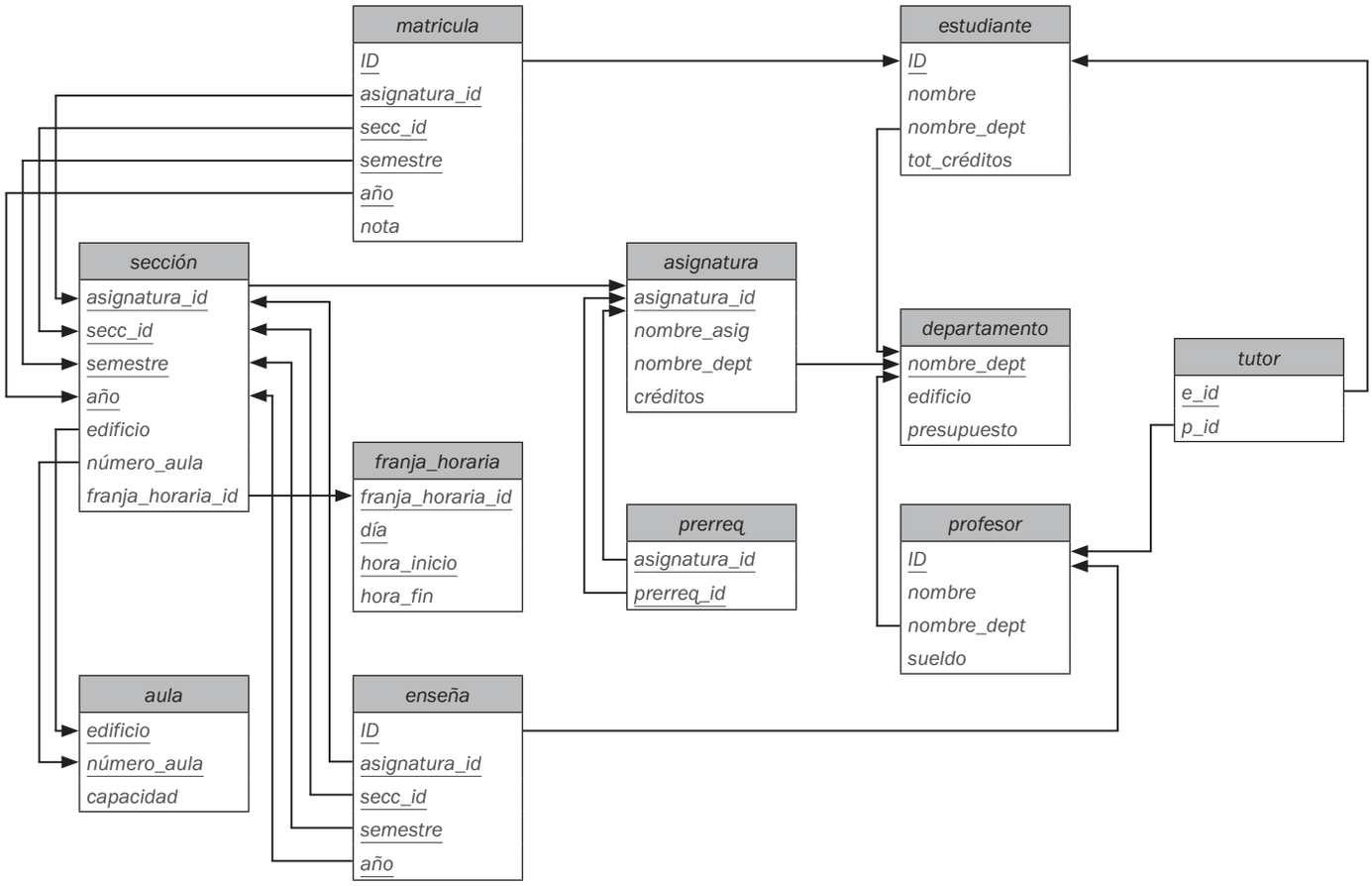


Figura 2.8. Diagrama de esquema para la base de datos de la universidad.

2.4. Diagramas de esquema

El esquema de la base de datos, junto con las dependencias de clave primaria y externa, se puede mostrar gráficamente mediante **diagramas de esquema.** La Figura 2.8 muestra el diagrama de esquema de nuestra universidad. Cada relación aparece como un cuadro con el nombre de la relación en la parte superior en un recuadro gris y los atributos en el interior del recuadro blanco. Los atributos que son clave primaria se muestran subrayados. Las dependencias de clave externa aparecen como flechas desde los atributos de clave externa de la relación referenciante a la clave primaria de la relación referenciada.

Las restricciones de integridad referencial distintas de las restricciones de clave externa no se muestran explícitamente en los diagramas de esquema. Se estudiará una representación gráfica diferente denominada diagrama entidad-relación más adelantes en el Capítulo 7. Los diagramas entidad-relación permiten representar distintos tipos de restricciones, incluyendo las restricciones de integridad referencial general.

Muchos sistemas de bases de datos proporcionan herramientas de diseño con una interfaz gráfica de usuario para la creación de los diagramas de esquema. En el Capítulo 7 se tratarán en profundidad las representaciones diagramáticas de los esquemas.

El ejemplo que su utilizará en los próximos capítulos es una universidad. La Figura 2.9 representa el esquema relacional que se utilizará en el ejemplo, en el que los atributos de clave primaria aparecen subrayados. Como se verá en el Capítulo 3, se corresponde con el enfoque de definir las relaciones en el lenguaje de definición de datos de SQL.

2.5. Lenguajes de consulta relacional

Un **lenguaje de consulta** es un lenguaje en el que los usuarios solicitan información de la base de datos. Estos lenguajes suelen ser de un nivel superior al de los lenguajes de programación habituales. Los lenguajes de consultas pueden clasificarse como procedimentales o no procedimentales. En los **lenguajes procedimentales,** el usuario indica al sistema que lleve a cabo una serie de operaciones en la base de datos para calcular el resultado deseado. En los **lenguajes no procedimentales,** el usuario describe la información deseada sin establecer un procedimiento concreto para obtener esa información.

aula(edificio, número_aula, capacidad)

departamento(nombre_dept, edificio, presupuesto)

asignatura(asignatura_id, nombre_asig, nombre_dept, créditos)

profesor(ID, nombre, nombre_dept, sueldo)

sección(asignatura_id, secc_id, semestre, año, edificio, número_aula, franja_horaria_id)

enseña(ID, asignatura_id, secc_id, semestre, año)

estudiante(ID, nombre, nombre_dept, tot_créditos)

matricula(ID, asignatura_id, secc_id, semestre, año, nota)

tutor(e_ID, p_ID)

franja_horaria(franja_horaria_id, día, hora_inicio, hora_fin)

prerreq(asignatura_id, prerreq_id)

Figura 2.9. Esquema de la base de datos de la universidad.

La mayor parte de los sistemas comerciales de bases de datos relacionales ofrecen un lenguaje de consultas que incluye elementos de los enfoques procedimental y no procedimental. Se estudiará ampliamente el lenguaje de consultas SQL en los Capítulos 3 al 5.

Existen varios lenguajes de consultas «puros»: el álgebra relacional es procedimental, mientras que el cálculo relacional de tuplas y el cálculo relacional de dominios no lo son. Estos lenguajes de consultas son rígidos y formales, y carecen del «azúcar sintáctico» de los lenguajes comerciales, pero ilustran las técnicas fundamentales para la extracción de datos de las bases de datos. En el Capítulo 6 se examina con detalle el álgebra relacional y las dos versiones de cálculo relacional, el cálculo relacional de tuplas y el cálculo relacional de dominios. El álgebra relacional consta de un conjunto de operaciones que toma una o dos relaciones como entrada y genera una nueva relación como resultado. El cálculo relacional usa la lógica de predicados para definir el resultado deseado sin proporcionar ningún procedimiento concreto para obtener dicho resultado.

2.6. Operaciones relacionales

Todos los lenguajes procedimentales de consulta relacional proporcionan un conjunto de operaciones que se pueden aplicar a una única relación o a un par de relaciones. Estas operaciones tienen la propiedad, por otra parte deseable, de que el resultado es siempre una única relación. Esta propiedad permite combinar varias de las operaciones de forma modular. Concretamente, como el resultado de una consulta relacional es una relación, se pueden aplicar operaciones a los resultados de las consultas de la misma forma que a los conjuntos de relaciones dadas.

Las operaciones relacionales concretas se expresan de forma diferente dependiendo del lenguaje, pero caen en el marco general que se describe en esta sección. En el Capítulo 3 se muestra la forma concreta de expresar estas operaciones en SQL.

La operación más frecuente es la selección de tuplas concretas de una única relación (por ejemplo, *profesor*) que satisfaga algún predicado particular (por ejemplo, *sueldo* > 85.000 €). El resultado es una nueva relación que sea un subconjunto de la relación original (*profesor*). Por ejemplo, si se seleccionan las tuplas de la relación *profesor* de la Figura 2.1, que satisfacen el predicado «*sueldo* es mayor que 85.000 €», se obtiene el resultado que se muestra en la Figura 2.10.

ID	*nombre*	*nombre_dept*	*sueldo*
12121	Wu	Finanzas	90000
22222	Einstein	Física	95000
33456	Gold	Física	87000
83821	Brandt	Informática	92000

Figura 2.10. Resultado de la consultas de seleccionar las tuplas de *profesor* con un sueldo superior a 85.000 €.

Otra operación habitual es seleccionar ciertos atributos (columnas) de una relación. El resultado es una nueva relación que solo tiene los atributos seleccionados. Por ejemplo, suponga que se desea una lista de los *ID* de los profesores y los sueldos sin listar los valores de *nombre* ni *nombre_dept* de la relación *profesor* de la Figura 2.1, entonces el resultado (Figura 2.11) posee los dos atributos, *ID* y *sueldo*. Las tuplas del resultado provienen de las tuplas de la relación *profesor* pero solo con los atributos que se muestran.

La operación ***join*** **(reunión)** permite combinar dos relaciones mezclando pares de tuplas, una de cada relación en una única tupla. Existen diferentes formas de unir relaciones (se verán en el Capítulo 3). En la Figura 2.12 se muestra un ejemplo de reunión de tuplas de las tablas *profesor* y *departamento* en la que las nuevas tuplas muestran la información de los profesores y los departamentos donde trabajan. El resultado se ha formado combinando las tuplas de la relación *profesor* con la tupla de la relación *departamento* por el departamento del profesor.

En la forma de reunión que se muestra en la Figura 2.12, que se denomina ***reunión natural,*** una tupla de la relación *profesor* coincide con una tupla de la relación *departamento* si los valores de sus atributos *nombre_dept* son iguales. Todas las tuplas así coincidentes son las que aparecen como resultado de la reunión. En general, la operación de reunión natural sobre dos relaciones hace casar las tuplas cuyos valores coinciden para el o los atributos que son comunes a ambas relaciones.

ID	*sueldo*
10101	65000
12121	90000
15151	40000
22222	95000
32343	60000
33456	87000
45565	75000
58583	62000
76543	80000
76766	72000
83821	92000
98345	80000

Figura 2.11. Resultado de la consulta de seleccionar los atributos *ID* y *sueldo* de la relación *profesor*.

ID	*nombre*	*sueldo*	*nombre_dept*	*edificio*	*presupuesto*
10101	Srinivasan	65000	Informática	Taylor	100000
12121	Wu	90000	Finanzas	Painter	120000
15151	Mozart	40000	Música	Packard	80000
22222	Einstein	95000	Física	Watson	70000
32343	El Said	60000	Historia	Painter	50000
33456	Gold	87000	Física	Watson	70000
45565	Katz	75000	Informática	Taylor	100000
58583	Califieri	62000	Historia	Painter	50000
76543	Singh	80000	Finanzas	Painter	120000
76766	Crick	72000	Biología	Watson	90000
83821	Brandt	92000	Informática	Taylor	100000
98345	Kim	80000	Electrónica	Taylor	85000

Figura 2.12. Resultado de la reunión natural de las relaciones *profesor* y *departamento*.

La operación ***producto cartesiano*** combina tuplas de dos relaciones, pero al contrario que la operación reunión, su resultado contiene *todos* los pares de tuplas de las dos relaciones independientemente de si sus atributos coinciden o no.

Como las relaciones son conjuntos, se pueden realizar operaciones de conjuntos sobre ellas. La operación ***unión*** realiza la unión de conjuntos de dos tablas «de estructura similar» (digamos una tabla de estudiantes graduados y una tabla de estudiantes aún sin graduar). Por ejemplo, se puede obtener el conjunto de todos los estudiantes de un departamento. También se pueden llevar a cabo otras operaciones sobre conjuntos, como la ***intersección*** y la ***diferencia de conjuntos.***

Como se ha indicado anteriormente, se pueden realizar operaciones sobre los resultados de las consultas. Por ejemplo, si se desea encontrar el *ID* y el *sueldo* de los profesores que tienen un sueldo de más de 85.000 €, se podrían utilizar las dos primeras operaciones del ejemplo anterior. En primer lugar se seleccionan las tuplas de la relación *profesor* en las que el valor *sueldo* es mayor de 85.000 € y, después del resultado, se seleccionan los dos atributos *ID* y *sueldo*, lo que genera como resultado la relación que se muestra en la Figura 2.13, que consta del *ID* y el *sueldo*. En este ejemplo, se podrían haber realizado las operaciones en cualquier orden, pero este no suele ser el caso en todas las situaciones, como se verá más adelante.

ID	*sueldo*
12121	90000
22222	95000
33456	87000
83821	92000

Figura 2.13. Resultado de seleccionar los atributos *ID* y *sueldo* de los profesores con un sueldo superior a 85.000 €.

A veces, el resultado de una consulta contiene tuplas duplicadas. Por ejemplo, si se selecciona el atributo *nombre_dept* de la relación *profesor*, existen varios casos de duplicados, incluyendo «Informática», que aparece tres veces. Ciertos lenguajes relacionales cumplen estrictamente con la definición matemática de conjunto y eliminan los duplicados. Otros, en consideración a la relativa sobrecarga de procesamiento necesario para eliminar los duplicados de relaciones muy grandes, mantienen los duplicados. En este último caso, las relaciones no son relaciones reales en el sentido matemático del término.

Por supuesto, los datos de la base de datos cambian con el tiempo. Una relación puede actualizarse insertando nuevas tuplas, borrando tuplas existentes o modificando tuplas cambiando valores de ciertos atributos. Se pueden borrar relaciones completas y crear otras nuevas.

Se tratarán las consultas y las actualizaciones relacionales usando el lenguaje SQL en los Capítulos 3 al 5.

ÁLGEBRA RELACIONAL

El álgebra relacional define un conjunto de operaciones sobre relaciones, de manera similar a las operaciones algebraicas habituales como la suma, la resta o la multiplicación que operan sobre números. Al igual que las operaciones algebraicas operan con uno o más números como entrada y devuelven un número como salida, el álgebra relacional normalmente opera con una o dos relaciones como entrada y devuelve una relación como salida.

El álgebra relacional se trata con detalle en el Capítulo 6, pero a continuación se describen algunas de sus operaciones:

Símbolo (Nombre)	Ejemplo de uso
σ (Selección)	$\sigma_{sueldo>=85000}(profesor)$
	Devuelve las filas de la relación de entrada que satisfacen el predicado.
Π (Proyección)	$\Pi_{ID,sueldo}(profesor)$
	Genera los atributos indicados de las filas de la relación de entrada. Elimina de la salida las tuplas duplicadas.
$\bowtie$ (Reunión natural)	$profesor \bowtie departamento$
	Genera pares de filas de las dos relaciones de entrada que tienen los mismos valores en todos los atributos con el mismo nombre.
$\times$ (Producto cartesiano)	$profesor \times departamento$
	Genera todos los pares de filas de las dos relaciones de entrada (independientemente de que tengan o no los mismos valores en el o los atributos comunes).
$\cup$ (Unión)	$\Pi_{nombre}(profesor) \cup \Pi_{nombre}(estudiante)$
	Genera la unión de las tuplas de las dos relaciones de entrada.

2.7. Resumen

- El **modelo de datos relacional** se basa en una colección de tablas. El usuario del sistema de bases de datos puede consultar estas tablas, insertar nuevas tuplas, borrar tuplas y actualizar (modificar) tuplas. Existen varios lenguajes para expresar estas operaciones.
- El **esquema** de una relación se refiere a su diseño lógico, mientras que un **ejemplar** de una relación se refiere a su contenido en un momento dado en el tiempo. El esquema de una base de datos y una instancia de una base de datos se definen de forma parecida. El esquema de una relación incluye sus atributos y, opcionalmente, los tipos de los atributos y las restricciones sobre las relaciones, como las restricciones de clave primaria y de clave externa.
- Una **superclave** de una relación es un conjunto de uno o más atributos cuyos valores se garantiza que identifican las tuplas de una relación de forma única. Una clave candidata es una superclave mínima, es decir, un conjunto de atributos que forman una superclave, pero que ninguno de sus subconjuntos es una superclave. Una de las claves candidatas de una relación se elige como su **clave primaria.**
- Una **clave externa** es un conjunto de atributos en una relación referenciante, de manera que para las tuplas de la relación referenciada los valores de los atributos de la clave externa se garantiza que existen como valores de la clave primaria en la relación referenciada.
- Un **diagrama de esquema** es una representación gráfica del esquema de una base de datos que muestra las relaciones de la base de datos, sus atributos, sus claves primarias y sus claves externas.
- Los **lenguajes de consulta relacional** definen un conjunto de operaciones que operan sobre tablas y generan tablas como resultado. Estas operaciones se pueden combinar para obtener expresiones que reflejen las consultas deseadas.
- El **álgebra relacional** proporciona un conjunto de operaciones que toman una o más relaciones como entrada y devuelven una relación como salida. Los lenguajes de consulta habituales, como SQL, se basan en el álgebra relacional, pero añaden un cierto número de características sintácticas de utilidad.

Términos de repaso

- Tabla.
- Relación.
- Tupla.
- Atributo.
- Dominio.
- Dominio atómico.
- Valor nulo.
- Esquema de la base de datos.
- Ejemplar de la base de datos.
- Esquema de relación.
- Ejemplar de relación.
- Claves.
  - Superclave.
  - Clave candidata.
  - Clave primaria.
- Clave externa.
  - Relación referenciante.
  - Relación referenciada.
- Restricción de integridad referencial.
- Diagrama de esquema.
- Lenguaje de consulta.
  - Lenguaje procedimental.
  - Lenguaje no procedimental.
- Operaciones sobre relaciones.
  - Selección de tuplas.
  - Selección de atributos.
  - Reunión natural.
  - Producto cartesiano.
  - Operaciones de conjuntos.
- Álgebra relacional.

Ejercicios prácticos

2.1. Considere la base de datos relacional de la Figura 2.14.
Indique las claves primarias apropiadas.

2.2. Considere las restricciones de clave externa del atributo *nombre_dept* de la relación *profesor* a la relación *departamento*.
Indique un ejemplo de inserciones y borrados de estas relaciones que genere un no cumplimiento de las restricciones de clave externa.

2.3. Considere la relación *franja_horaria*. Dado que una franja horaria concreta puede darse más de una vez a la semana, explique por qué *día* y *hora_inicio* forman parte de la clave primaria de esta relación, mientras que *hora_fin* no.

2.4. En el ejemplar de *profesor* que se muestra en la Figura 2.1 no existen dos profesores con el mismo nombre. De esto, ¿se puede concluir que el nombre se puede utilizar como superclave (o clave primaria) de profesor?

2.5. Indique el resultado de realizar primero el producto cartesiano de *estudiante* y *tutor* y después realizar la operación de selección sobre el resultado con el predicado $e_id = ID$. Usando la notación simbólica del álgebra relacional, esta consulta se escribe como $\sigma_{e_id=ID}(estudiante \times tutor)$.

empleado (nombre_persona, calle, ciudad)

trabaja (nombre_persona, nombre_empresa, sueldo)

empresa (nombre_empresa, ciudad)

Figura 2.14. Base de datos relacional para los Ejercicios 2.1, 2.7 y 2.12.

sucursal (nombre_sucursal, ciudad_sucursal, activos)

cliente (nombre_cliente, calle_cliente, ciudad_cliente)

préstamo (número_préstamo, nombre_sucursal, cantidad)

prestatario (nombre_cliente, número_préstamo)

cuenta (número_cuenta, nombre_sucursal, saldo)

impositor (nombre_cliente, número_cuenta)

Figura 2.15. Base de datos del banco para los Ejercicios 2.8, 2.9 y 2.13.

2.6. Considere las siguientes expresiones, que utilizan el resultado de una operación del álgebra relacional como entrada de otra operación. Para cada expresión, describa con palabras qué operación realiza.

a. $\sigma_{año \geq 2009}(matricula) \bowtie estudiante$

b. $\sigma_{año \geq 2009}(matricula \bowtie estudiante)$

c. $\Pi_{ID,nombre,asignatura_id}(estudiante \bowtie matricula)$

2.7. Considere la base de datos relacional de la Figura 2.14. Indique una expresión del álgebra relacional que exprese cada una de las siguientes consultas:

a. Encontrar los nombres de todos los empleados que viven en la ciudad de «Miami».

b. Encontrar los nombres de todos los empleados cuyos sueldos sean superiores a 100.000 €.

c. Encontrar los nombre de todos los empleados que vivan en «Miami» y cuyos sueldos sean superiores a 100.000 €.

2.8. Considere la base de datos del banco de la Figura 2.15. Indique una expresión del álgebra relacional para cada una de las siguientes consultas:

a. Encontrar los nombres de todas las sucursales ubicadas en «Chicago».

b. Encontrar los nombres de todos los prestatarios que tienen un préstamo en la sucursal «Downtown».

Ejercicios

2.9. Considere la base de datos del banco de la Figura 2.15.

a. Indique cuáles son las claves primarias apropiadas.

b. Dada su elección de claves primarias, identifique las claves externas apropiadas.

2.10. Considere la relación *tutor* que se muestra en la Figura 2.8, con *e_id* como clave primaria de *tutor*. Suponga que un estudiante puede tener más de un tutor. Entonces, ¿seguiría sirviendo *e_id* como clave primaria de la relación *tutor*? Si su respuesta es no, ¿cuál debería ser la cave primaria de *tutor*?

2.11. Describa la diferencia de significado entre los términos *relación* y *esquema de relación*.

2.12. Considere la base de datos de la Figura 2.14. Indique una expresión del álgebra relacional para expresar las siguientes consultas:

a. Encontrar los nombre de todos los empleados que trabajan para «First Bank Corporation».

b. Encontrar los nombres y las ciudades de residencia de todos los empleados que trabajan para «First Bank Corporation».

c. Encontrar los nombres, la calle y la ciudad de residencia de todos los empleados que trabajan para «First Bank Corporation» y ganan más de 10.000 €.

2.13. Considere la base de datos del banco de la Figura 2.15. Escriba una expresión del álgebra relacional para cada una de las siguientes consultas:

a. Encontrar todos los números de préstamos con un valor de préstamo superior a 10.000 €.

b. Encontrar los nombre de todos los impositores que tienen una cuenta con un valor superior a 6.000 €.

c. Encontrar los nombre de todos los impositores que tienen una cuenta con un valor superior a 6.000 € en la sucursal «Uptown».

2.14. Indique dos razones por las que se deben introducir los valores nulos en la base de datos.

2.15. Discuta los méritos relativos de los lenguajes procedimentales y los no procedimentales.

Notas bibliográficas

E. F. Codd, del Laboratorio San Jose Research de IBM, propuso el modelo relacional a finales de los años sesenta (Codd [1970]). Este trabajo le permitió a Codd obtener el prestigioso premio ACM Turing Award en 1981 (Codd [1982]).

Tras la publicación por Codd de su original artículo, se formaron varios proyectos de investigación para construir sistemas de bases de datos relacionales prácticas, incluyendo el Sistema R en el Laboratorio San Jose Research de IBM, Ingres en la Universidad de California, en Berkeley, y Query-by-Example en el Centro de investigación T. J. Watson de IBM.

Ahora existen en el mercado muchos productos de bases de datos relacionales. Entre ellos se encuentran DB2 de IBM e Informx, Oracle, Sybase y SQL Server de Microsoft.

Entre los sistemas de bases de datos relaciones de fuente abierta están MySQL y PostgreSQL. Microsoft Access es un producto de bases de datos de un solo usuario que forma parte de Microsoft Office.

Atzeni y Antonellis [1993], Maier [1983] y Abiteboul et ál. [1995] son textos dedicados exclusivamente a la teoría de los modelos de datos relacionales.

Introducción a SQL

03

Existen varios lenguajes de consulta de bases de datos, tanto comerciales como experimentales. En este capítulo y en los Capítulos 4 y 5 se estudiará con detalle el lenguaje SQL, el más ampliamente utilizado.

Aunque nos referimos al lenguaje SQL como un «lenguaje de consulta», puede hacer mucho más que realizar consultas a la base de datos. Puede definir estructuras, modificar los datos y especificar restricciones de seguridad.

No es la intención de este libro proporcionar una guía de usuario de SQL completa, sino que presentaremos los elementos y conceptos fundamentales de SQL. Las distintas implementaciones de SQL pueden modificar distintos detalles o pueden proporcionar solamente un subconjunto de todo el lenguaje.

3.1. Introducción al lenguaje de consultas SQL

IBM desarrolló la versión original de SQL, en un primer momento denominada Sequel, como parte del proyecto System R, a principios de 1970. El lenguaje Sequel ha evolucionado desde entonces y su nombre ha pasado a ser SQL (Structured Query Language, lenguaje estructurado de consultas). Hoy en día, numerosos productos son compatibles con el lenguaje SQL y se ha establecido como *el* lenguaje estándar para las bases de datos relacionales.

En 1986, ANSI (American National Standards Institute o Instituto estadounidense de normalización) e ISO (International Standards Organization u Organización internacional de normalización) publicaron una norma SQL, denominada SQL-86. En 1989, ANSI publicó una extensión de la norma para SQL denominada SQL-89. La siguiente versión de la norma fue SQL-92, seguida de SQL:1999; SQL:2003, SQL:2006 y la versión más reciente, SQL:2008. Las notas bibliográficas proporcionan referencias a esas normas.

El lenguaje SQL tiene varios componentes:

- **Lenguaje de definición de datos (LDD).** El LDD de SQL proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificación de los esquemas de relación.
- **Lenguaje interactivo de manipulación de datos (LMD).** El LMD de SQL incluye un lenguaje de consultas como comandos para insertar, borrar y modificar tuplas en la base de datos.
- **Integridad.** El LDD de SQL incluye comandos para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos. Las actualizaciones que violan las restricciones de integridad se rechazan.
- **Definición de vistas.** El LDD de SQL incluye comandos para la definición de vistas.
- **Control de transacciones.** SQL incluye comandos para especificar el comienzo y el final de las transacciones.
- **SQL incorporado** y **SQL dinámico.** SQL incorporado y SQL dinámico definen cómo se pueden incorporar instrucciones de SQL en lenguajes de programación de propósito general como C, C++ y Java.
- **Autorización.** El LDD de SQL incluye comandos para especificar los derechos de acceso a las relaciones y a las vistas.

En este capítulo se presenta una visión general del LMD básico y de las características esenciales del LDD de SQL. Esta descripción se basa principalmente en la muy extendida norma SQL-92.

En el Capítulo 4 se ofrece un tratamiento más detallado del lenguaje SQL, incluyendo: (a) varias expresiones de unión; (b) vistas; (c) transacciones; (d) restricciones de integridad; (e) sistema de tipos y (f) autorización.

En el Capítulo 5 trataremos las características más avanzadas del lenguaje SQL, entre ellas: (a) mecanismos para utilizar SQL desde lenguajes de programación; (b) funciones y procedimientos de SQL; (c) disparadores; (d) consultas recursivas; (e) características avanzadas de agregación y (f) distintas características diseñadas para el análisis de datos, que se introdujeron en SQL:1999 y las versiones posteriores. Más adelante, en el Capítulo 22, se tratarán las extensiones orientadas a objetos, introducidas en SQL:1999.

Aunque la mayoría de las implementaciones de SQL disponen de las características del estándar que describiremos, debe tener cuidado, ya que existen diferencias entre las distintas implementaciones. La mayoría disponen de elementos no estándar, mientras que otras omiten algunas de las características más avanzadas. En el caso de que se encuentre que algunas de las características que describimos no funcionan en el lenguaje de consulta que utiliza normalmente, consulte el manual de usuario de su sistema de base de datos para ver exactamente qué características están disponibles.

3.2. Definición de datos de SQL

El conjunto de relaciones de cada base de datos debe especificarse en el sistema en términos de un lenguaje de definición de datos (LDD). El LDD de SQL no solo permite la especificación de un conjunto de relaciones, sino también de la información relativa a esas relaciones, incluyendo:

- El esquema de cada relación.
- El dominio de valores asociado a cada atributo.
- Las restricciones de integridad.
- El conjunto de índices que se deben mantener para cada relación.
- La información de seguridad y de autorización de cada relación.
- La estructura de almacenamiento físico de cada relación en el disco.

A continuación se verá la definición de esquemas y tipos básicos; más adelante se tratarán otras características avanzadas del LDD de SQL, en los Capítulos 4 y 5.

3.2.1. Tipos básicos

La norma SQL soporta gran variedad de tipos de dominio predefinidos, entre ellos:

- **char***(n)*. Una cadena de caracteres de longitud fija, con una longitud *n* especificada por el usuario. También se puede utilizar la palabra completa **character.**
- **varchar***(n)*. Una cadena de caracteres de longitud variable con una longitud máxima *n* especificada por el usuario. La forma completa, **character varying,** es equivalente.
- **int.** Un entero (un subconjunto finito de los enteros dependiente de la máquina). La palabra completa, **integer,** es equivalente.
- **smallint.** Un entero pequeño (un subconjunto dependiente de la máquina del tipo de dominio entero).
- **numeric***(p, d)*. Un número de coma fija, cuya precisión la especifica el usuario. El número está formado por *p* dígitos (más el signo), y de esos *p* dígitos, *d* pertenecen a la parte decimal. Así, **numeric**(3,1) permite que el número 44.5 se almacene exactamente, pero ni 444.5 ni 0.32 se pueden almacenar exactamente en un campo de este tipo.
- **real, double precision.** Números de coma flotante y números de coma flotante de doble precisión, con precisión dependiente de la máquina.
- **float***(n)*. Un número de coma flotante cuya precisión es, al menos, de *n* dígitos.

En la Sección 4.5 se tratan otros tipos adicionales.

Todos los tipos incluyen un valor especial llamado valor ***null*** **(nulo).** Un valor nulo indica una ausencia de valor que puede existir o no pero que puede no ser conocido. En algunos casos es posible que se desee prohibir que se puedan introducir valores nulos, como se verá en breve.

El tipo de dato **char** almacena una cadena de caracteres de tamaño fijo. Suponga, por ejemplo, un atributo *A* del tipo **char**(10). Si guardamos la cadena «Avi» en este atributo, se añaden 7 espacios para que tenga un total de 10 caracteres de tamaño. Por otro lado, suponga un atributo *B* del tipo **varchar**(10), donde guardamos «Avi»; en este atributo no se añaden espacios extra. Cuando se comparan dos valores de tipo **char,** si tienen distinto tamaño se añaden automáticamente espacios extra al más pequeño para que tengan el mismo tamaño antes de realizar la comparación.

Cuando se compara un tipo **char** con un tipo varchar, uno esperaría que se añadiesen espacios extra al tipo **varchar** para que tengan el mismo tamaño antes de la comparación; sin embargo, puede que se haga o no, dependiendo del sistema de bases de datos. El resultado final es que aunque se guarde el mismo valor «Avi», en los atributos *A* y *B* como se ha indicado, la comparación *A=B* puede devolver falso. Se recomienda que siempre se utilice el tipo **varchar** en lugar del tipo **char** para evitar estos problemas.

SQL también proporciona el tipo **nvarchar** para guardar datos en varios idiomas utilizando la representación Unicode. Sin embargo, muchas bases de datos permiten guardar Unicode incluso en tipos **varchar** (en la representación UTF-8).

3.2.2. Definición básica de esquemas

Las relaciones se definen mediante el comando **create table.** El siguiente comando crea una relación *departamento* en la base de datos:

create table *departamento*
(*nombre_dept* **varchar** (20),
edificio **varchar** (15),
presupuesto **numeric** (12,2),
primary key (*nombre_dept*));

La relación creada anteriormente tiene tres atributos *nombre_dept*, que es una cadena de caracteres de un tamaño máximo de 20; *edificio*, que es una cadena de caracteres de un tamaño máximo de 15 y *presupuesto*, que es un número con un total de 12 dígitos, de los cuales 2 están tras el punto decimal. El comando **create table** también indica que el atributo *nombre_dept* es la clave primaria de la relación *departamento*.

La forma general del comando **create table** es:

create table *r*
(A_1 D_1,
A_2 D_2,
…,
A_n D_n,
⟨restricción-integridad$_1$⟩,
…,
⟨restricción-integridad$_k$⟩);

donde *r* es el nombre de la relación, cada una de las A_i es el nombre de los atributos del esquema de la relación *r*, y D_i es el dominio del atributo A_i ; es decir, D_i indica el tipo del atributo A_i junto con las restricciones de integridad opcionales que restringen el conjunto de valores permitidos para A_i.

El punto y coma que se muestra al final de las sentencias **create table,** así como al final de otras sentencias de SQL que se verán más adelante en este capítulo, en la mayoría de las implementaciones de SQL son opcionales.

SQL admite un número variable de restricciones de integridad. En esta sección solo se tratarán algunas de ellas:

- **primary key** (A_{j1}, A_{j2}, …, A_{jm}): la especificación de **clave primaria** determina que los atributos A_{j1}, A_{j2}, …, A_{jm} forman la clave primaria de la relación. Los atributos de la clave primaria tienen que ser *no nulos* y *únicos*; es decir, ninguna tupla puede tener un valor nulo para un atributo de la clave primaria y ningún par de tuplas de la relación puede ser igual en todos los atributos de la clave primaria. Aunque la especificación de clave primaria es opcional, suele ser una buena idea especificar una clave primaria para cada relación.
- **clave externa** (A_{k1}, A_{k2}, …, A_{kn}) **references** *s*: la **clave externa** indica que los valores de los atributos (A_{k1}, A_{k2}, …, A_{kn}) de cualquier tupla en la relación se deben corresponder con los valores de los atributos de la clave primaria de otras tuplas de la relación *s*.

  En la Figura 3.1 se presenta una definición parcial del LDD de SQL de la base de datos de la universidad que se usa en el libro. La definición de la tabla *asignatura* tiene una declaración «**foreign key** (*nombre_dept*) **references** *departamento*». Esta declaración indica que en las tuplas de asignatura, el nombre de departamento indicado en la tupla debe existir en el atributo de clave primaria (*nombre_dept*) de la relación *departamento*. Sin esta restricción, sería posible que una asignatura indicase un nombre de departamento que no existiese. En la Figura 3.1 también se muestran restricciones de clave externa en las tablas *sección*, *profesor* y *enseña*.
- **not null:** la restricción **not null** en un atributo indica que no se permite el valor nulo para ese atributo; en otras palabras, la restricción excluye el valor nulo de entre los valores del dominio de ese atributo. Por ejemplo, en la Figura 3.1 la restricción **not null** en el atributo *nombre* de la relación *profesor* asegura que el nombre del profesor no puede ser nulo.

Otras restricciones de clave externa, así como otras restricciones de integridad que puede incluir el comando **create table** se tratan más adelante, en la Sección 4.4.

```
create table departamento
   (nombre_dept      varchar (20),
   edificio          varchar (15),
   presupuesto       numeric (12,2),
   primary key (nombre_dept));

create table asignatura
   (asignatura_id    varchar (7),
   nombre            varchar (50),
   nombre_dept       varchar (20),
   créditos          numeric (2,0),
   primary key (asignatura_id),
   foreign key (nombre_dept) references departamento);

create table profesor
   (ID               varchar (5),
   nombre            varchar (20) not null,
   nombre_dept       varchar (20),
   sueldo            numeric (8,2),
   primary key (ID),
   foreign key (nombre_dept) references departamento);

create table sección
   (asignatura_id    varchar (8),
   secc_id           varchar (8),
   semestre          varchar (6),
   año               numeric (4,0),
   edificio          varchar (15),
   número_aula       varchar (7),
   franja_horaria_id varchar (4),
   primary key (asignatura_id, secc_id, semestre, año),
   foreign key (asignatura_id) references asignatura);

create table enseña
   (ID               varchar (5),
   asignatura_id     varchar (8),
   secc_id           varchar (8),
   semestre          varchar (6),
   año               numeric (4,0),
   primary key (ID, asignatura_id, secc_id, semestre, año),
   foreign key (asignatura_id, secc_id, semestre, año) references sección,
   foreign key (ID) references profesor);
```

Figura 3.1. Definición de datos SQL de parte de la base de datos de la universidad.

En SQL se evita que cualquier actualización de la base de datos viole las restricciones de integridad de los atributos de clave primaria o, si la tupla tiene los mismos valores en la clave primaria que otra de la tuplas, SQL indica un error y evita dicha actualización. De forma similar, durante una inserción de una tupla *asignatura* con un valor de *nombre_dept* que no exista en la relación *departamento*, que generaría una violación de la restricción de clave externa de *asignatura*, SQL impediría realizar tal inserción.

Todas las relaciones recién creadas están inicialmente vacías. Se utiliza el comando **insert** para introducir datos en la relación. Por ejemplo, si se desea introducir el hecho de que existe un profesor de nombre Smith en el departamento de Biología con un *profesor_id* 10211 y un sueldo de 66.000 €, se escribirá:

```
insert into profesor
   values (10211, ´Smith´, ´Biología´, 66000);
```

Los valores se indican en el *orden* en que se encuentran los atributos en el esquema de relación. El comando insert tiene otras funciones de utilidad que se tratarán con más detalle en la Sección 3.9.2.

Se puede utilizar el comando **delete** para eliminar tuplas de una relación. El comando:

delete from *estudiante*;

elimina todas las tuplas de la relación *estudiante*. Otras formas del comando delete (borrar) permiten indicar tuplas concretas para eliminarlas; el comando delete se tratará con detalle más adelante en la Sección 3.9.1.

Para eliminar una relación de una base de datos SQL se utiliza el comando **drop table.** Este comando elimina toda la información de la relación indicada de la base de datos. El comando:

drop table *r*;

es una acción mucho más drástica que:

delete from *r*;

Esta última mantiene la relación *r*, solo elimina las tuplas de *r*. La primera elimina no solo las tuplas de *r*, sino también el esquema de *r*. Una vez que se ha utilizado el comando **drop table** no se pueden volver a insertar tuplas en *r*, a no ser que se vuelva a crear la relación con el comando **create table.**

Se utiliza el comando **alter table** para añadir atributos a una relación ya existente. Todas las tuplas de la relación tendrán el valor *null* en los nuevos atributos añadidos. El comando **alter table** tiene la forma:

alter table *r* **add** *A D*;

donde *r* es el nombre de la relación existente, *A* es el nombre del atributo a añadir y *D* es el tipo del atributo añadido. Se pueden eliminar atributos de una relación utilizando el comando:

alter table *r* **drop** *A*;

donde *r* es el nombre de una relación que ya existe, y *A* es el nombre de uno de los atributos de dicha relación. Muchos sistemas de bases de datos no admiten eliminar atributos, aunque sí se puede eliminar una tabla completa.

3.3. Estructura básica de las consultas SQL

La estructura básica de una consulta SQL consta de tres cláusulas: **select, from** y **where.** Las consultas tienen como entrada las relaciones indicadas en la cláusula **from,** operan con ellas como se indica en las cláusulas **where** y **select** y generan como resultado una relación. Se verá la sintaxis SQL mediante ejemplos y se describirá la estructura general de las consultas SQL más adelante.

3.3.1. Consultas sobre una relación única

Supongamos una consulta simple sobre nuestro ejemplo de universidad, «encontrar los nombres de todos los profesores». Los nombres de los profesores se encuentran en la relación *profesor*, por tanto se indica la relación en la cláusula **from.** El nombre del profesor se encuentra en el atributo *nombre*, por lo que se indica en la cláusula **select.**

select *nombre*
from *profesor*;

El resultado es una relación que consta de un único atributo con el encabezado *nombre*. Si la relación *profesor* es la que se mostraba en la Figura 2.1, entonces la relación resultado de la consulta anterior es la que se muestra en la Figura 3.2.

A continuación suponga la consulta «encontrar los nombres de departamento de todos los profesores, que se puede escribir»:

select *nombre_dept*
from *profesor*;

Como puede haber más de un profesor perteneciente a un departamento, el nombre de departamento puede aparecer más de una vez en la relación *profesor*. El resultado de la consulta anterior es una relación que contiene los nombres de departamento, como se muestra en la Figura 3.3.

Según la definición matemática, formal, del modelo de relación, una relación es un conjunto. Por tanto, nunca deberían aparecer tuplas repetidas en una relación. En la práctica, la eliminación de duplicados es muy costosa. Por tanto, en SQL se permiten duplicados en las relaciones, así como en los resultados de expresiones SQL. Por eso en la consulta SQL anterior aparece el departamento una vez por cada tupla que exista en la relación *profesor*.

En aquellos casos en los que se desee forzar a que no existan duplicados, se añade la palabra clave **distinct** tras **select.** Se puede escribir la consulta anterior como:

select distinct *nombre_dept*
from *profesor*;

si deseamos eliminar los duplicados. El resultado de la consulta anterior contendría cada nombre de departamento solo una vez.

SQL nos permite utilizar la palabra clave **all** para indicar explícitamente que los duplicados no se eliminen:

select all *nombre_dept*
from *profesor*;

Como por defecto se mantienen todos los duplicados, en el resto del ejemplo no se utilizará la palabra **all.** Para asegurarnos que se eliminan los duplicados en el resto de ejemplos se incluirá **distinct,** siempre que sea necesario.

La cláusula **select** también puede contener expresiones aritméticas que incluyan los operadores $+$, $-$, $*$ y $/$ con constantes o atributos de las tuplas. Por ejemplo, la consulta:

select *ID*, *nombre*, *nombre_dept*, *sueldo** 1.1
from *profesor*;

devuelve una relación que es la misma que la relación *profesor* excepto que el atributo *sueldo* se multiplica por 1.1. Sería como aumentar el sueldo un 10 por ciento a todos los profesores; fíjese, sin embargo, que no se produce ningún cambio en la relación *profesor*.

SQL también proporciona tipos de datos especiales, como varias formas del tipo *fecha*, y permite que se puedan utilizar operadores aritméticos con estos tipos. Se tratarán más adelante en la Sección 4.5.1.

La cláusula **where** permite seleccionar solamente aquellas filas de la relación resultado de la cláusula **from** que satisfagan determinado predicado. Suponga la consulta «encontrar los nombres de todos los profesores del departamento de Informática cuyo sueldo sea superior a 70.000 €». Esta consulta se puede escribir en SQL como:

select *nombre*
from *profesor*
where *nombre_dept* = 'Informática' **and** *sueldo* > 70000;

Dada la relación *profesor* que se muestra en la Figura 2.1, entonces la relación resultado de la consulta anterior es la que se muestra en la Figura 3.4.

SQL permite el uso de operadores lógicos **and, or** y **not** en la cláusula **where.** Los operandos de los operadores lógicos pueden ser expresiones que incluyan los operadores de comparación $<$, $<=$, $>$, $>=$, $=$ y $<>$. SQL permite utilizar los operadores de comparación para comparar cadenas de caracteres y expresiones aritméticas, así como otros tipos especiales, como los tipos fecha (*date*).

Se tratarán otras características de la cláusula **where** más adelante en este capítulo.

nombre
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Figura 3.2. Resultado de «**select** *nombre* **from** *profesor*».

nombre_dept
Informática
Finanzas
Música
Física
Historia
Física
Informática
Historia
Finanzas
Biología
Informática
Electrónica

Figura 3.3. Resultado de «**select** *nombre_dept* **from** *profesor*».

nombre
Katz
Brandt

Figura 3.4. Resultado de «encontrar los nombres de todos los profesores del departamento de Informática cuyo sueldo sea superior a 70.000 €».

3.3.2. Consultas sobre varias relaciones

Hasta ahora las consultas de los ejemplos eran sobre una sola relación, pero lo habitual es que se necesite acceder a información de varias relaciones. A continuación se tratará cómo escribir estas consultas.

A modo de ejemplo, suponga que se desea responder a la siguiente consulta: «obtener los nombres de todos los profesores, junto con los nombres de sus departamentos y el nombre del edificio donde se encuentra el departamento».

Si observa el esquema de la relación *profesor*, se puede ver que es posible obtener el nombre del departamento del atributo *nombre_dept*, pero el nombre del edificio del departamento se encuentra en el atributo *edificio* de la relación *departamento*. Para resolver la consulta, las tuplas de la relación *profesor* hay que casarlas con las tuplas de la relación *departamento* cuyo valor *nombre_dept* coincida con el valor de *nombre_dept* de la tupla *profesor*.

En SQL, para responder a la consulta anterior se indican las relaciones a las que se necesita acceder en la cláusula **from** y se especifica la condición de coincidencia en la cláusula **where.** La consulta anterior se puede escribir en SQL como:

select *nombre*, *profesor.nombre_dept*, *edificio*
from *profesor*, *departamento*
where *profesor.nombre_dept* = *departamento.nombre_dept*;

Si las relaciones *profesor* y *departamento* son las que se muestran en las Figuras 2.1 y 2.5, respectivamente, el resultado de esta consulta es el que se muestra en la Figura 3.5.

Fíjese que el atributo *nombre_dept* existe tanto en la relación *profesor* como en *departamento*, y el nombre de la relación se usa como un prefijo (en *profesor.nombre_dept*, y en *departamento.nombre_dept*) para indicar claramente a qué atributo se refiere. Al contrario, los atributos *nombre* y *edificio* aparecen solo en una de las relaciones y, por tanto, no hay que indicar como prefijo el nombre de la relación.

Este convenio de denominación *requiere* que las relaciones que se indican en la cláusula **from** tengan nombres diferentes. Semejante requisito ocasiona problemas en algunos casos, como cuando se necesita combinar la información de dos tuplas diferentes en la misma relación. En la Sección 3.4.1 se verá cómo evitar estos problemas mediante la operación **rename.**

A continuación se considerará el caso general de las consultas SQL que implican a varias relaciones. Como se ha tratado anteriormente, una consulta SQL puede contener tres tipos de cláusulas, la cláusula **select,** la cláusula **from** y la cláusula **where.** El papel que juega cada una de ellas es el siguiente:

- La cláusula **select** se utiliza para indicar los atributos deseados en el resultado de una consulta.
- La cláusula **from** indica la lista de relaciones a las que hay que acceder para evaluar la consulta.
- La cláusula **where** es un predicado que incluye atributos de la relación de la cláusula **from.**

Una consulta típica de SQL tiene la forma:

select $A_1, A_2, \ldots, A_n$
from $r_1, r_2 \ldots, r_m$
where P;

Donde A_i representa un atributo, los r_i una relación y P es un predicado. Si se omite la cláusula **where,** el predicado P es **true.**

Aunque haya que escribir las cláusulas en el orden **select, from, where,** la forma más sencilla de comprender las operaciones que se especifican en una consulta es considerarlas en el orden: **from, where** y **select.**[1]

La cláusula **from** se define por sí misma como el producto cartesiano de las relaciones indicadas. Formalmente se define en términos de la teoría de conjuntos, pero la mejor forma de entenderlo es como un proceso iterativo que genera tuplas de la relación resultado de la cláusula **from.**

for each tupla t_1 **in** relación r_1
  for each tupla t_2 **in** relación r_2
    …
    for each tupla t_m **in** relación r_m
      Concatenar $t_1, t_2, \ldots, t_m$ en una sola tupla t
      Añadir t a la relación resultado

1 En la práctica, SQL puede convertir la expresión en una forma equivalente que se pueda procesar de manera más eficiente. Sin embargo, se dejarán los temas de eficiencia para los Capítulos 12 y 13.

nombre	*nombre_dept*	*edificio*
Srinivasan	Informática	Taylor
Wu	Finanzas	Painter
Mozart	Música	Packard
Einstein	Física	Watson
El Said	Historia	Painter
Gold	Física	Watson
Katz	Informática	Taylor
Califieri	Historia	Painter
Singh	Finanzas	Painter
Crick	Biología	Watson
Brandt	Informática	Taylor
Kim	Electrónica	Taylor

Figura 3.5. Resultado de «obtener los nombres de todos los profesores, junto con los nombres de sus departamentos y el nombre del edificio donde se encuentra el departamento».

La relación resultado tiene todos los atributos de todas las relaciones de la cláusula **from.** Como puede aparecer el mismo nombre de atributo tanto en r_i como en r_j, como ya se ha indicado anteriormente, se pone un prefijo del nombre de la relación original antes del nombre del atributo.

Por ejemplo, el esquema de relación para el producto cartesiano de las relaciones *profesor* y *enseña* es:

(*profesor.ID*, *profesor.nombre*, *profesor.nombre_dept*, *profesor.sueldo enseña.ID*, *enseña.asignatura_id*, *enseña.secc_id*, *enseña.semestre*, *enseña.año*)

Con este esquema se puede distinguir entre *profesor.ID* y *enseña.ID*. En los atributos que aparecen solo una vez en cada uno de los esquemas normalmente se elimina el prefijo nombre de la relación. Esta simplificación no genera ninguna ambigüedad. Se puede escribir el esquema de la relación como:

(*profesor.ID*, *nombre*, *nombre_dept*, *sueldo enseña.ID*, *asignatura_id*, *secc_id*, *semestre*, *año*)

Considere ahora la relación *profesor* de la Figura 2.1 y la relación *enseña* de la Figura 2.7. Su producto cartesiano es el que se muestra en la Figura 3.6, que incluye solo una parte de las tuplas que constituyen el resultado del producto cartesiano.

El producto cartesiano combina las tuplas de *profesor* y *enseña* aunque no haya relación entre ellas. Cada tupla de *profesor* se combina con *cada* tupla de *enseña*, incluso aquellas que se refieren a profesores diferentes. El resultado puede ser una relación muy extensa, y en raras ocasiones tiene sentido crear tal producto cartesiano.

En su lugar, se utiliza el predicado de la cláusula **where** para restringir la combinación que crea el producto cartesiano a aquellas que tienen sentido para la respuesta deseada. Cabría esperar que una consulta que implique a *profesor* y *enseña* combinará una tupla t de *profesor* solo con aquellas tuplas de *enseña* que se refieren a ese mismo profesor. Es decir, solo son deseables las tuplas de *enseña* con las tuplas de *profesor* que tengan el mismo valor de *ID*. La siguiente consulta de SQL hace cumplir esta condición y da como resultado los nombres de los profesores y los identificadores de asignatura de las tuplas que coinciden:

select *nombre*, *asignatura_id*
from *profesor*, *enseña*
where *profesor.ID* = *enseña.ID*;

profesor.ID	*nombre*	*nombre_dept*	*sueldo*	*enseña.ID*	*asignatura_id*	*secc_id*	*semestre*	*año*
10101	Srinivasan	Física	95000	10101	CS-101	1	Otoño	2009
10101	Srinivasan	Física	95000	10101	CS-315	1	Primavera	2010
10101	Srinivasan	Física	95000	10101	CS-347	1	Otoño	2009
10101	Srinivasan	Física	95000	10101	FIN-201	1	Primavera	2010
10101	Srinivasan	Física	95000	15151	MU-199	1	Primavera	2010
10101	Srinivasan	Física	95000	22222	PHY-101	1	Otoño	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Física	95000	10101	CS-101	1	Otoño	2009
12121	Wu	Física	95000	10101	CS-315	1	Primavera	2010
12121	Wu	Física	95000	10101	CS-347	1	Otoño	2009
12121	Wu	Física	95000	10101	FIN-201	1	Primavera	2010
12121	Wu	Física	95000	15151	MU-199	1	Primavera	2010
12121	Wu	Física	95000	22222	PHY-101	1	Otoño	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Física	95000	10101	CS-101	1	Otoño	2009
15151	Mozart	Física	95000	10101	CS-315	1	Primavera	2010
15151	Mozart	Física	95000	10101	CS-347	1	Otoño	2009
15151	Mozart	Física	95000	10101	FIN-201	1	Primavera	2010
15151	Mozart	Física	95000	15151	MU-199	1	Primavera	2010
15151	Mozart	Física	95000	22222	PHY-101	1	Otoño	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Física	95000	10101	CS-101	1	Otoño	2009
22222	Einstein	Física	95000	10101	CS-315	1	Primavera	2010
22222	Einstein	Física	95000	10101	CS-347	1	Otoño	2009
22222	Einstein	Física	95000	10101	FIN-201	1	Primavera	2010
22222	Einstein	Física	95000	15151	MU-199	1	Primavera	2010
22222	Einstein	Física	95000	22222	PHY-101	1	Otoño	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Figura 3.6. Producto cartesiano de la relación *profesor* con la relación *enseña*.

Fíjese que la consulta anterior solo tiene como resultado profesores que han enseñado alguna asignatura. Los profesores que no han enseñado ninguna asignatura no se muestran; si se desea obtener estas tuplas, podríamos usar la operación llamada *reunión externa* (*outer join*), que se describe en la Sección 4.1.2.

Si la relación *profesor* es la que se muestra en la Figura 2.1 y la relación *enseña* es la que se muestra en la Figura 2.7, entonces el resultado de la consulta anterior es la que se muestra en la Figura 3.7. Observe que los profesores Gold, Califieri y Singh, que no han enseñado ninguna asignatura, no aparecen entre los resultados.

Si quisiéramos encontrar los nombres de los profesores y los identificadores de asignatura de los profesores del departamento de Informática, se podría añadir un predicado extra a la cláusula **where** de la siguiente forma:

select *nombre*, *asignatura_id*
from *profesor*, *enseña*
where *profesor.ID* = *enseña.ID* **and**
profesor.nombre_dept = 'Informática';

Fíjese en que como el atributo *nombre_dept* solo aparece en la relación *profesor*, se podría haber usado solo *nombre_dept*, en lugar de *profesor.nombre_dept*, en la consulta anterior.

En general, el significado de una consulta en SQL se puede entender de la siguiente forma:

1. Genere un producto cartesiano de las relaciones de la cláusula **from.**
2. Aplique los predicados indicados en la cláusula **where** al resultado del paso 1.
3. Para cada tupla del paso 2, extraiga los atributos (o resultado de las expresiones) indicados en la cláusula **select.**

nombre	*asignatura_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

Figura 3.7. Resultado de «De los profesores de la universidad que hayan enseñado alguna asignatura, encontrar el nombre y la asignatura_id de todas las asignaturas que se han enseñado».

La secuencia de pasos anterior indica claramente cuál debería ser el resultado de una consulta SQL, *no* cómo se debería ejecutar. Una implementación real de SQL no debería ejecutar la consulta de esta forma; debería optimizar la evaluación generando (hasta donde sea posible) solo los elementos del producto cartesiano que satisfagan los predicados de la cláusula **where.** Se tratarán las técnicas de implementación más adelante en los Capítulos 12 y 13.

Cuando se escriben las consultas se debería ser cuidadoso a la hora de incluir las condiciones apropiadas de la cláusula **where.** Si se omitiesen las condiciones de la cláusula **where** en la consulta anterior, daría como resultado el producto cartesiano, que podría ser una relación de gran tamaño. Del ejemplo de la relación *profesor* de la Figura 2.1 y el ejemplo de la relación *enseña* de la Figura 2.7, su producto cartesiano tiene $12 * 13 = 156$ tuplas – más de las que podemos mostrar en el texto. Aún peor, suponga un número más realista de profesores que el de la relación del ejemplo, digamos que 200 profesores. Supongamos que cada profesor enseña 3 asignaturas, por lo que tendremos 600 tuplas en la relación *enseña*. Entonces, el proceso iterativo anterior genera $200 * 600 = 120.000$ tuplas en el resultado.

3.3.3. La unión natural

En nuestra consulta de ejemplo que combinaba la información de las tablas *profesor* y *enseña*, la condición de coincidencia era que *profesor.ID* fuese igual a *enseña.ID*. Estos son los únicos atributos de las dos relaciones que tienen el mismo nombre. De hecho es un caso habitual, es decir, la condición de coincidencia de la cláusula **from** suele requerir que todos los atributos con los mismos nombres sean iguales.

Para facilitar las cosas a los programadores para un caso tan común como este, en SQL se dispone de la llamada *unión natural*, que se describe a continuación. De hecho, SQL dispone de muchas otras formas para poder **reunir** la información de dos o más relaciones. Ya se ha visto cómo el producto cartesiano junto con el predicado de la cláusula **where** se pueden utilizar para reunir la información de varias relaciones. En la Sección 4.1 se tratan otras formas de reunir información de varias relaciones.

La operación **unión natural** opera con dos relaciones y genera como resultado una relación. Al contrario que el producto cartesiano de dos relaciones, que concatena todas las tuplas de la primera relación con cada una de las tuplas de la segunda, la unión natural solo considera aquellos pares de tuplas con los mismos valores en los atributos que aparecen en los esquemas de ambas relaciones. Por tanto, volviendo al ejemplo de las relaciones *profesor* y *enseña*, el cálculo de *profesor* **natural join** *enseña* solo considera aquellos pares de tuplas en los que tanto las tuplas de *profesor* como las tuplas de *enseña* tienen los mismos valores en el atributo común *ID*.

La relación resultado, como se muestra en la Figura 3.8, solo tiene 13 tuplas, las que dan información sobre un profesor y una asignatura que enseña dicho profesor. Tenga en cuenta que no se repiten los atributos que aparecen en los esquemas de ambas relaciones, sino que solamente aparecen una vez. Fíjese también en el orden en que aparecen los atributos, primero los comunes a los esquemas de ambas relaciones, después los atributos únicos al esquema de la primera relación y, finalmente, los atributos únicos al esquema de la segunda relación.

Supóngase la consulta «Para todos los profesores de la universidad que hayan enseñado alguna asignatura, encontrar su nombre y la asignatura_id de todas las asignaturas que hayan enseñado», que anteriormente ya escribimos como:

select *nombre*, *asignatura_id*
from *profesor*, *enseña*
where *profesor.ID* = *enseña.ID*;

Esta consulta se puede escribir de forma más concisa usando la unión natural de SQL como:

select *nombre*, *asignatura_id*
from *profesor* **natural join** *enseña*;

Las dos consultas anteriores generan el mismo resultado.

Como ya se ha visto anteriormente, el resultado de la operación unión natural es una relación. Conceptualmente, la expresión «*profesor* **natural join** *enseña*» en la cláusula **from** se sustituye por la relación obtenida de la evaluación de la unión natural.[2] Las cláusulas **where** y **select** se evalúan después en esta relación, como ya se vio en la Sección 3.3.2.

Una cláusula **from** en una consulta SQL puede tener muchas relaciones combinadas utilizando la unión natural, como se muestra a continuación:

select $A_1, A_2, \ldots, A_n$
from r_1 **natural join** r_2 **natural join** … **natural join** r_m
where P;

2 Como consecuencia, no se pueden utilizar nombres de atributos que contengan los nombres originales de las relaciones, por ejemplo, *profesor.nombre* o *enseña.asignatura_id*, para referirse a los atributos del resultado de la unión natural; sin embargo, se pueden utilizar los nombres de atributos como *nombre* y *asignatura_id*, sin el nombre de la relación.

ID	*nombre*	*nombre_dept*	*sueldo*	*asignatura_id*	*secc_id*	*semestre*	*año*
10101	Srinivasan	Informática	65000	CS-101	1	Otoño	2009
10101	Srinivasan	Informática	65000	CS-315	1	Primavera	2010
10101	Srinivasan	Informática	65000	CS-347	1	Otoño	2009
12121	Wu	Finanzas	90000	FIN-201	1	Primavera	2010
15151	Mozart	Música	40000	MU-199	1	Primavera	2010
22222	Einstein	Física	95000	PHY-101	1	Otoño	2009
32343	El Said	Historia	60000	HIS-351	1	Primavera	2010
45565	Katz	Informática	75000	CS-101	1	Primavera	2010
45565	Katz	Informática	75000	CS-319	1	Primavera	2010
76766	Crick	Biología	72000	BIO-101	1	Verano	2009
76766	Crick	Biología	72000	BIO-301	1	Verano	2010
83821	Brandt	Informática	92000	CS-190	1	Primavera	2009
83821	Brandt	Informática	92000	CS-190	2	Primavera	2009
83821	Brandt	Informática	92000	CS-319	2	Primavera	2010
98345	Kim	Electrónica	80000	EE-181	1	Primavera	2009

Figura 3.8. Unión natural de la relación *profesor* con la relación *enseña*.

De forma más general, una cláusula **from** puede ser de la forma:

from $E_1, E_2, \ldots, E_n$

donde cada E_i puede ser una sola relación o una expresión con uniones naturales. Por ejemplo, suponga que se desea responder a la consulta «listar los nombres de los profesores junto con los nombres de las asignaturas que enseñan».

La consulta se puede escribir en SQL como:

select *nombre*, *nombre_asig*
from *profesor* **natural join** *enseña*, *asignatura*
where *enseña.asignatura_id* = *asignatura.asignatura_id*;

Primero se calcula la unión natural de *profesor* y *enseña*, como se ha visto anteriormente, y se calcula el producto cartesiano de este resultado con *asignatura*, de donde la cláusula **where** extrae solo aquellas tuplas en las que el identificador de asignatura del resultado de la unión coincide con el identificador de asignatura de la relación *asignatura*. Fíjese que *enseña.asignatura_id* en la cláusula **where** se refiere al campo *asignatura_id* del resultado de la unión natural, ya que este campo proviene de la relación *enseña*.

Sin embargo, la siguiente consulta SQL *no* calcula el mismo resultado:

select *nombre*, *nombre_asig*
from *profesor* **natural join** *enseña* **natural join** *asignatura*;

Para ver por qué, fíjese en que la unión natural de *profesor* y *enseña* contiene los atributos (*ID*, *nombre*, *nombre_dept*, *sueldo*, *asignatura_id*, *secc_id*), mientras que la relación *asignatura* contiene los atributos (*asignatura_id*, *nombre_asig*, *nombre_dept*, *créditos*). El resultado es que la unión natural de estas dos requeriría que los valores del atributo *nombre_dept* de las dos entradas fuesen el mismo, además de requerir que los valores de *asignatura_id* fuesen los mismos.

Esta consulta dejaría fuera todos los pares (nombre profesor, nombre asignatura) en los que el profesor enseña una asignatura en un departamento distinto al propio del profesor. La consulta previa, por otra parte, generaba correctamente dichos pares.

Para obtener las ventajas de la unión natural evitando el peligro de la igualdad errónea entre atributos, SQL proporciona una forma del constructor unión natural que permite indicar exactamente qué columnas se deberían igualar. Esta característica se muestra en la siguiente consulta:

select *nombre*, *nombre_asig*
from (*profesor* **natural join** *enseña*) **join** *asignatura*
  using (*asignatura_id*);

La operación **join**... **using** requiere que se indique una lista de nombres de atributos. Hay que indicar los nombres de los atributos de las dos relaciones de entrada.

Considere la operación r_1 **join** r_2 **utilizando** (A_1, A_2). La operación es similar a r_1 **natural join** r_2, excepto que un par de tuplas t_1 de r_1 y t_2 de r_2 casan si $t_1.A_1 = t_2.A_1$ y $t_1.A_2 = t_2.A_2$; incluso si r_1 y r_2 tienen un atributo de nombre A_3, *no* se requiere que $t_1.A_3 = t_2.A_3$.

Por tanto, en la consulta anterior en SQL el constructor **join** permite que *enseña.nombre_dept* y *asignatura.nombre_dept* difieran, y la consulta SQL genera la respuesta correcta.

3.4. Operaciones básicas adicionales

Existen otras operaciones básicas adicionales definidas en SQL que pasamos a ver en los siguientes apartados.

3.4.1. La operación de renombrado

Suponga de nuevo la consulta que se ha utilizado anteriormente:

select *nombre*, *asignatura_id*
from *profesor*, *enseña*
where *profesor.ID* = *enseña.ID*;

El resultado de esta consulta es una relación con los siguientes atributos:

nombre, *asignatura_id*

Los nombres de los atributos en el resultado provienen de los nombres de los atributos en las relaciones de la cláusula **from.**

Sin embargo, no se pueden derivar siempre los nombres de esta forma, por varias razones. En primer lugar, las dos relaciones de la cláusula **from** pueden tener atributos con los mismos nombres, en cuyo caso uno de los nombres de atributo se encuentra duplicado en el resultado. En segundo lugar, si se utiliza una expresión aritmética en la cláusula **select,** el atributo resultado no tiene nombre. En tercer lugar, aunque se pueda deducir el nombre del atributo de la relación base como en el ejemplo anterior, puede que queramos cambiar este nombre en el resultado. Así que SQL proporciona un mecanismo de renombrado de atributos de la relación resultado. Se utiliza la cláusula **as,** de la forma:

nombre_anterior **as** *nombre_nuevo*

La cláusula **as** puede aparecer tanto en la cláusula **select** como en la cláusula **from.**[3]

Por ejemplo, si se desea sustituir el nombre del atributo *nombre* por el nombre *nombre_profesor*, se puede rescribir la consulta anterior como:

select *nombre* **as** *nombre_profesor*, *asignatura_id*
from *profesor*, *enseña*
where *profesor.ID* = *enseña.ID*;

La cláusula **as** resulta particularmente útil en el renombrado de relaciones. Una razón para renombrar una relación es sustituir un nombre de la relación muy largo por una versión más corta que se pueda usar de forma más sencilla en la consulta. Para verlo, se reescribirá la consulta «Para todos los profesores de la universidad que hayan enseñado alguna asignatura, encontrar sus nombres y las asignatura_id de todas las asignaturas que hayan enseñado»:

select *T.nombre*, *S.asignatura_id*
from *profesor* **as** *T*, *enseña* **as** *S*
where *T.ID* = *S.ID*;

Otra razón para renombrar una relación es el caso en el que se desean comparar tuplas de la misma relación. Entonces se necesita realizar el producto cartesiano de una relación consigo misma, y sin el renombrado, resulta imposible distinguir una tupla de la otra. Suponga que se desea escribir la consulta «Encontrar los nombres de todos los profesores cuyo sueldo sea mayor que al menos un profesor del departamento de Biología». Se puede escribir la expresión SQL:

select distinct *T.nombre*
from *profesor* **as** *T*, *profesor* **as** *S*
where *T.sueldo* > *S.sueldo* **and** *S.nombre_dept* = ´Biología´;

3 Versiones previas de SQL no incluían la palabra clave **as.** Por ello, algunas implementaciones de SQL, como Oracle, no permitían la palabra **as** en la cláusula **from.** En Oracle «*nombre_anterior* **as** *nombre_nuevo*» se escribe como «*nombre_anterior nombre_nuevo*» en la cláusula **from.** La palabra clave **as** se permite para el renombrado de atributos en la cláusula **select,** pero es opcional, y en Oracle se puede omitir.

Observe que no podríamos utilizar la notación *profesor.sueldo*, ya que no queda claro a qué referencia *profesor* atañe.

En la consulta anterior se dice que T y S son copias de la relación *profesor*, pero para ser más preciso, se declaran como alias, que son nombres alternativos para la relación *profesor*. El identificador, como T o S, que se utiliza para el renombrado de una relación se denomina **nombre de correlación** en la norma de SQL, pero normalmente se llaman **alias de tabla,** o **variable de correlación,** o **variable de tupla.**

Una forma más clara de leer la consulta anterior en español sería «Encontrar los nombres de todos los profesores que ganan más que el profesor con menor sueldo del departamento de Biología». La frase original casa mejor con lo escrito en SQL, pero esta otra resulta más intuitiva y, de hecho, se puede expresar directamente en SQL, como se verá en la Sección 3.8.2.

3.4.2. Operaciones con cadenas de caracteres

SQL especifica las cadenas de caracteres encerrándolas entre comillas simples, como en ´Ordenador´, como ya se ha visto anteriormente. Los caracteres de comillas que formen parte de una cadena de caracteres se pueden especificar usando dos caracteres de comillas; por ejemplo, la cadena de caracteres «Peter O´Toole» se puede especificar como «Peter O´´Toole».

La norma SQL especifica que la operación de igualdad entre cadenas de caracteres es sensible a las mayúsculas, por lo que la expresión «´información´ = ´Información´» se evalúa a falso. Sin embargo, algunas bases de datos, como MySQL y SQL Server, no distinguen entre mayúsculas y minúsculas cuando se comparan cadenas de caracteres, por lo que «´información´ = ´Información´» siempre se evaluará a cierto en estas bases de datos. Este comportamiento por defecto se puede cambiar, bien en el nivel de la base de datos o bien en el nivel de los atributos concretos.

SQL también permite varias funciones que operan sobre cadenas de caracteres, tales como la concatenación (utilizando «||»), la extracción de subcadenas de caracteres, el cálculo de la longitud de las cadenas de caracteres, la conversión a mayúsculas (utilizando **upper**(s)) y minúsculas (utilizando **lower**(s)), la eliminación de espacios al final de las cadenas (utilizando using **trim**(s)). También hay variaciones en los distintos conjuntos de funciones sobre cadenas de caracteres que ofrecen los distintos sistemas de bases de datos. Consulte el manual del sistema de su base de datos para obtener los detalles sobre las funciones que admite.

También se puede realizar la comparación de patrones utilizando el operador **like.** Los patrones se escriben utilizando caracteres especiales:

- Tanto por ciento (%). El carácter % coincide con cualquier subcadena de caracteres.
- Subrayado (_). El carácter _ coincide con cualquier carácter.

Los patrones distinguen entre mayúsculas y minúsculas; es decir, los caracteres en mayúscula se consideran diferentes de los caracteres en minúscula, y viceversa. Para ilustrar la comparación de patrones se considerarán los siguientes ejemplos:

- ´Intro%´ coincide con cualquier cadena de caracteres que empiece con «Intro».
- ´%Infor%´ coincide con cualquier cadena que contenga «Infor» como subcadena; por ejemplo, ´Intro. a la Informática´ e ´Informática biológica´.
- ´_ _ _´ coincide con cualquier cadena que tenga exactamente tres caracteres.
- ´_ _ _%´ coincide con cualquier cadena que tenga, al menos, tres caracteres.

Los patrones se expresan en SQL utilizando el operador de comparación **like.** Considérese la consulta «Encontrar los nombres de todos los departamentos cuyo nombre de edificio incluye la subcadena ´Watson´». Esta consulta se puede escribir como:

select *nombre_dept*
from *departamento*
where *edificio* **like** ´%Watson%´;

Para que los patrones puedan contener los caracteres especiales de patrón (esto es, % y _) SQL permite la especificación de un carácter de escape. El carácter de escape se utiliza inmediatamente antes de los caracteres especiales de patrón para indicar que ese carácter especial se va a tratar como un carácter normal. El carácter de escape para una comparación **like** se define utilizando la palabra clave **escape.** Para ilustrar esto, considérense los siguientes patrones, que utilizan una barra invertida (\) como carácter de escape:

- **like** ´ab\%cd%´ **escape** ´\´ coincide con todas las cadenas que empiecen por «ab%cd».
- **like** ´ab\\cd%´ **escape** ´\´ coincide con todas las cadenas que empiecen por «ab\cd».

SQL permite buscar discordancias en lugar de concordancias utilizando el operador de comparación **not like.** Algunas bases de datos proporcionan variantes de la operación **like** que no distinguen entre mayúsculas y minúsculas.

SQL:1999 también ofrece una operación **similar to,** que proporciona una comparación de patrones más potente que la operación **like;** la sintaxis para especificar patrones es parecida a la usada para las expresiones regulares de Unix.

3.4.3. Especificación de atributos en la cláusula Select

El símbolo asterisco «*» se puede utilizar en la cláusula **select** para indicar «todos los atributos». Por tanto, el uso de *profesor*.* en la cláusula **select** de la consulta:

select *profesor*.*
from *profesor*, *enseña*
where *profesor.ID* = *enseña.ID*;

indica que se seleccionan todos los atributos de *profesor*. Una cláusula **select** de la forma **select*** indica que se seleccionan todos los atributos de la relación resultado de la cláusula **from.**

3.4.4. Orden en la presentación de las tuplas

SQL ofrece al usuario cierto control sobre el orden en el cual se presentan las tuplas de una relación. La cláusula **order by** hace que las tuplas del resultado de una consulta se presenten en un cierto orden. Para obtener una relación en orden alfabético de todos los profesores del departamento de Física se escribe:

select *nombre*
from *profesor*
where *nombre_dept* = ´Física´
order by *nombre*;

De manera predeterminada, la cláusula **order by** coloca los elementos en orden ascendente. Para especificar el tipo de ordenación se puede especificar **desc** para orden descendente o **asc** para orden ascendente. Además, se puede ordenar con respecto a más de un atributo. Suponga que se desea listar todos los *profesores* en orden descendente de *sueldo*. Si varios profesores tienen el mismo sueldo, se ordenan en orden ascendente de nombre. Se expresa del modo siguiente:

select*
from *profesor*
order by *sueldo* **desc,** *nombre* **asc;**

3.4.5. Predicados de la cláusula where

SQL incluye el operador de comparación **between** para simplificar las cláusulas **where** que indican que un valor sea menor o igual que algún valor, y mayor o igual que otro. Si se desea encontrar los nombres de los profesores con sueldos entre 90.000 € y 100.000 €, se puede utilizar la comparación **between** para escribir:

select *nombre*
from *profesor*
where *sueldo* **between** 90000 **and** 100000;

en lugar de:

select *nombre*
from *profesor*
where *sueldo* <= 100000 **and** *sueldo* >= 90000;

De forma similar se puede utilizar el operador de comparación **not between.**

Se puede extender la consulta anterior para encontrar los nombres de los profesores junto con los identificadores de las asignaturas, que ya se ha visto anteriormente, y considerar un caso más complicado en el que además se quiera que los profesores sean del departamento de Biología: «Encontrar los nombres de los profesores y las asignaturas que enseñan para todos los profesores del departamento de Biología que hayan enseñado alguna asignatura». Para escribir esta consulta se puede modificar la consulta SQL ya vista añadiendo una condición extra en la cláusula **where.** Esta sería una versión modificada de la consulta SQL que usa la unión natural:

select *nombre*, *asignatura_id*
from *profesor*, *enseña*
where *profesor.ID* = *enseña.ID* **and** *nombre_dept* = ´Biología´;

SQL nos permite utilizar la notación $(v_1, v_2, \ldots, v_n)$ para indicar una tupla de cardinalidad n que contiene los valores $v_1, v_2, \ldots, v_n$. Se pueden utilizar los operadores de comparación y el orden se define de forma lexicográfica. Por ejemplo, $(a_1, a_2) <= (b_1, b_2)$ es cierto si $a_1 <= b_1$ **and** $a_2 <= b_2$; de forma similar, las dos tuplas son iguales si todos sus atributos son iguales. Por tanto, la consulta SQL anterior se puede escribir como:[4]

select *nombre*, *asignatura_id*
from *profesor*, *enseña*
where (*profesor.ID*, *nombre_dept*) = (*enseña.ID*, ´Biología´);

asignatura_id
CS-101
CS-347
PHY-101

Figura 3.9. Relación *c1*, con el listado de asignaturas enseñadas en el otoño de 2009.

3.5. Operaciones sobre conjuntos

Las operaciones de SQL **union, intersect** y **except** operan sobre relaciones y se corresponden con las operaciones matemáticas de la teoría de conjuntos ∪, ∩ y −. A continuación se van a construir consultas que impliquen las operaciones **union, intersect** y **except** con dos conjuntos.

4 Aunque forme parte del estándar SQL-92, puede que algunas implementaciones de SQL no admitan esta sintaxis.

- El conjunto de todas las asignaturas que se enseñan en el semestre de otoño de 2009:

  select *asignatura_id*
  from *sección*
  where *semestre* = ´Otoño´ **and** *año* = 2009;

- El conjunto de todas las asignaturas que se enseñan en el semestre de primavera de 2010:

  select *asignatura_id*
  from *sección*
  where *semestre* = ´Primavera´ **and** *año* = 2010;

A partir de ahora se hará referencia a las relaciones obtenidas como resultado de las consultas anteriores como *c1* y *c2*, respectivamente, y se mostrarán los resultados de la ejecución de estas consultas sobre la relación *sección* de la Figura 2.6, en las Figuras 3.9 y 3.10. Observe que *c2* contiene dos tuplas que se corresponden con *asignatura_id* CS-319, ya que se ofertaron dos secciones de la asignatura en la primavera de 2010.

asignatura_id
CS-101
CS-315
CS-319
CS-319
FIN-201
HIS-351
MU-199

Figura 3.10. La relación *c2* lista las asignaturas que se enseñaron en la primavera de 2010.

3.5.1. La operación unión

Para encontrar todas las asignaturas que se enseñaron en el otoño de 2009 o en la primavera de 2010, o en ambas, se escribe:[5]

(**select** *asignatura_id*
from *sección*
where *semestre* = ´Otoño´ **and** *año* = 2009)
union
(**select** *asignatura_id*
from *sección*
where *semestre* = ´Primavera´ **and** *año* = 2010);

A diferencia de la cláusula **select,** la operación **union** (unión) elimina los valores duplicados automáticamente. Así, usando la relación sección de la Figura 2.6, en la que hay dos secciones de la asignatura CS-319 que se ofrecieron en la primavera de 2010, y una sección de CS-101 que se ofreció en el otoño de 2009, así como en el semestre de otoño de 2010, las asignaturas CS-101 y CS-319 solo aparecen una vez en el resultado, tal como se muestra en la Figura 3.11.

Si se desea mantener todos los duplicados, hay que escribir **union all** en lugar de **union:**

(**select** *asignatura_id*
from *sección*
where *semestre* = ´Otoño´ **and** *año* = 2009)
union all
(**select** *asignatura_id*
from *sección*
where *semestre* = ´Primavera´ **and** *año* = 2010);

5 El paréntesis que se incluye alrededor de las sentencias **select-from-where** es opcional, pero facilita la lectura.

El número de tuplas duplicadas en el resultado es igual al número total de valores duplicados que aparecen en *c1* y *c2*. Así, en la consulta anterior tanto CS-319 como CS-101 aparecerían dos veces. Si fuese el caso de que se hubiesen enseñado cuatro secciones de ECE-101 en el semestre de otoño de 2009 y dos secciones de ECE-101 en el semestre de otoño de 2010, entonces en el resultado aparecerían seis tuplas de ECE-101.

asignatura_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Figura 3.11. Relación resultado de *c1* union *c2*.

3.5.2. La operación intersección

Para encontrar todas las asignaturas que se enseñaron en el otoño de 2009, así como en la primavera de 2010, escribimos:

(**select** *asignatura_id*
 from *sección*
 where *semestre* = ´Otoño´ **and** *año* = 2009)
intersect
(**select** *asignatura_id*
 from *sección*
 where *semestre* = ´Primavera´ **and** *año* = 2010);

La relación resultado, que se muestra en la Figura 3.12, solo contiene una tupla con la asignatura CS-101. La operacion **intersect** (intersección) elimina los valores duplicados automáticamente. Por ejemplo, si se diese el caso de que se enseñasen cuatro secciones de la asignatura ECE-101 en el semestre de otoño de 2009 y dos secciones de la asignatura ECE-101 en el semestre de primavera de 2010, en el resultado solo habría una tupla con ECE-101.

Si se quisieran mantener todos los duplicados habría que escribir **intersect all** en lugar de **intersect:**

(**select** *asignatura_id*
 from *sección*
 where *semestre* = ´Otoño´ **and** *año* = 2009)
intersect all
(**select** *asignatura_id*
 from *sección*
 where *semestre* = ´Primavera´ **and** *año* = 2010);

El número de tuplas duplicadas que aparece en el resultado es igual al número mínimo de valores duplicados que aparecen en *c1* y *c2*. Por ejemplo, si se diese el caso de que se enseñasen cuatro secciones de la asignatura ECE-101 en el semestre de otoño de 2009 y dos secciones de la asignatura ECE-101 en el semestre de primavera de 2010, entonces habría dos tuplas con ECE-101 en el resultado.

asignatura_id
CS-101

Figura 3.12. Relación resultado de *c1* intersect *c2*.

3.5.3. La operación excepto

Para encontrar todas las asignaturas que se enseñaron en el semestre de otoño de 2009 pero que no se enseñaron en el semestre de primavera de 2010, escribimos:

(**select** *asignatura_id*
 from *sección*
 where *semestre* = ´Otoño´ **and** *año* = 2009)
except
(**select** *asignatura_id*
 from *sección*
 where *semestre* = ´Primavera´ **and** *año* = 2010);

El resultado de esta consulta se muestra en la Figura 3.13. Fíjese en que es exactamente la relación *c1* de la Figura 3.9, excepto que la tupla de CS-101 no aparece. La operación **except**[6] da como salida todas las tuplas de su primera entrada que no existen en la segunda; es decir, realiza la diferencia. Por ejemplo, si se diese el caso de que se enseñasen cuatro secciones de la asignatura ECE-101 en el semestre de otoño de 2009 y dos secciones de la asignatura ECE-101 en el semestre de primavera de 2010, el resultado de la operación **except** no tendría ninguna copia de ECE-101.

Si se quisiesen mantener los duplicados, se debería utilizar **except all** en lugar de **except:**

(**select** *asignatura_id*
 from *sección*
 where *semestre* = ´Otoño´ **and** *año* = 2009)
except all
(**select** *asignatura_id*
 from *sección*
 where *semestre* = ´Primavera´ **and** *año* = 2010);

El número de copias duplicadas de una tupla en el resultado es igual al número de copias duplicadas en *c1* menos el número de copias duplicadas en *c2*, siempre que esta diferencia sea positiva. Por tanto, si se diese el caso de que se enseñasen cuatro secciones de la asignatura ECE-101 en el semestre de otoño de 2009 y dos secciones de la asignatura ECE-101 en el semestre de primavera de 2010, entonces habría dos tuplas con ECE-101 en el resultado. Sin embargo, si hubiese dos o menos secciones de ECE-101 en el semestre de otoño de 2009 y dos secciones en el semestre de primavera de 2010, no quedaría ninguna tupla en el resultado.

asignatura_id
CS-347
PHY-101

Figura 3.13. Relación resultado de *c1* except *c2*.

3.6. Valores nulos

El uso de valores *nulos* en las operaciones aritméticas, de comparación y de conjuntos causa algunos problemas especiales.

El resultado de las expresiones aritméticas (que incluyan por ejemplo $+$, $-$, $*$ o $/$) es nulo si cualquiera de los valores de entrada es nulo. Por ejemplo, si una consulta tiene una expresión $r.A + 5$, y $r.A$ es nulo para alguna de las tuplas, entonces la expresión debe tener también valor nulo para dicha tupla.

Las comparaciones que implican valores nulos también son un problema. Por ejemplo, suponga la comparación «$1 <$ **null**». Sería erróneo decir que es cierto ya que no se conoce qué valor representa *null*. Pero también sería erróneo decir que es falso; si lo hi-

6 Algunas implementaciones de SQL, en particular Oracle, utilizan la palabra **minus** en lugar de **except.**

ciésemos, «**not** (1 < **null**)» debería evaluarse como cierto, lo que tampoco tiene ningún sentido. Por tanto, SQL trata como **unknown** (desconocido) el resultado de cualquier comparación en la que intervenga el valor *null* (excepto para los predicados **is null** e **is not null,** que se describen más adelante). Ello crea un tercer valor lógico, además de *true* (verdadero) y *false* (falso).

Como el predicado en una cláusula **where** puede implicar valores booleanos en operaciones como **and, or** y **not** como resultado de las comparaciones, las definiciones de las operaciones booleanas deben extenderse para tratar con el valor **unknown.**

- **and:** el resultado de *true* **and** *unknown* es *unknown*, *false* **and** *unknown* es *false*, mientras que *unknown* **and** *unknown* es *unknown*.
- **or:** el resultado de *true* **or** *unknown* es *true*, *false* **or** *unknown* es *unknown*, mientras que *unknown* **or** *unknown* es *unknown*.
- **not:** el resultado de **not** *unknown* es *unknown*.

Se puede verificar que si *r.A* es *null*, entonces «1 < *r.A*» así como «**not** (1 < *r.A*)» se evalúan a *unknown*.

Si el predicado de la cláusula **where** se evalúa a **false** o a **unknown** para una de las tuplas, dicha tupla no se añade al resultado.

SQL utiliza la palabra especial **null** en un predicado para comprobar el valor *null*. Por tanto, para encontrar a todos los profesores que aparezcan en la relación *profesor* con valores *null* en el *sueldo*, escribimos:

select *nombre*
from *profesor*
where *sueldo* **is null;**

El predicado **is not null** es cierto si el valor al que se aplica no vale *null*.

Algunas implementaciones de SQL también permiten comprobar si el resultado de una comparación es desconocido (*unknown*), en lugar de *true* o *false*, utilizando las cláusulas **is unknown** e **is not unknown.**

Cuando una consulta utiliza la cláusula **select distinct,** se deben eliminar las tuplas duplicadas. Para ello, cuando se comparan los valores correspondientes a dos tuplas, estos se tratan como idénticos si ambos son distintos de *null* e iguales en valor, como si ambos son *null*. Por tanto, dos copias de una tupla, de la forma {(´A´,*null*), (´A´,*null*)}, se tratan como si fuesen idénticas, incluso aunque algunos de los atributos tengan valores *null*. Al usar la cláusula **distinct** solo se mantiene una copia de dichas tuplas idénticas. Fíjese que el tratamiento de los valores nulos en este caso es diferente del de los valores nulos en los predicados, en los que la comparación «*null* = *null*» devolvería *unknown* (desconocido), en lugar de cierto.

Esta forma de tratar las tuplas como idénticas si tienen los mismos valores para todos los atributos, incluso si algunos de ellos fuesen *null*, también se utiliza para las operaciones de conjuntos: unión, intersección y excepto.

3.7. Funciones de agregación

Las *funciones de agregación* toman una colección de valores (un conjunto o multiconjunto) y devuelven un único valor. SQL dispone de cinco funciones de agregación incorporadas:

- Media: **avg.**
- Mínimo: **min.**
- Máximo: **max.**
- Total: **sum.**
- Recuento: **count.**

La entrada de **sum** y **avg** debe ser una colección de números, pero el resto de operadores pueden operar también con colecciones de tipos de datos no numéricos, como cadenas de caracteres.

3.7.1. Agregación básica

Considere la consulta: «Encontrar el sueldo medio de los profesores del departamento de Informática». La consulta se escribiría como:

select avg (*sueldo*)
from *profesor*
where *nombre_dept* = ´Informática´;

El resultado de esta consulta es una relación con un único atributo, que contiene una única tupla con un valor numérico correspondiente a la media del sueldo de todos los profesores del departamento de Informática. El sistema de base de datos puede dar un nombre arbitrario al atributo de la relación resultado que se genera de la agregación; sin embargo, podemos asignarle un nombre con sentido utilizando la cláusula **as,** como:

select avg (*sueldo*) **as** *med_sueldo*
from *profesor*
where *nombre_dept* = ´Informática´;

En la relación *profesor* de la Figura 2.1 los salarios del departamento de Informática son 75.000 €, 65.000 € y 92.000 €. El sueldo medio es de 232.000 €/3 = 77.333,33 €.

Mantener los duplicados es importante cuando se calcula una media. Suponga que se añade un cuarto profesor al departamento de Informática cuyo sueldo es de 75.000 €. Si se eliminasen los duplicados, se obtendría una respuesta errónea (232.000 €/4 = 58.000 €) en lugar de la respuesta correcta, que sería 76.750 €.

Hay casos en los que hay que eliminar los duplicados antes de calcular una función de agregación. Si no se desea eliminar los duplicados, se usa la palabra clave **distinct** en la expresión de agregación. Un ejemplo se puede encontrar en la consulta «Encontrar el número total de profesores que enseñaron alguna asignatura en el semestre de primavera de 2010». En este caso, el profesor se contabiliza una sola vez, independientemente del número de asignaturas que haya enseñado. La información requerida se encuentra en la relación *enseña*, y esta consulta se escribe de la siguiente forma:

select count (**distinct** *ID*)
from *enseña*
where *semestre* = ´Primavera´ **and** *año* = 2010;

Como la palabra clave **distinct** precede a *ID*, aunque un profesor enseñe más de una asignatura solo se contabiliza una vez en el resultado.

Usaremos la función de agregación **count** muy a menudo para contar el número de tuplas de una relación. La notación para esta función en SQL es **count** (*). Por tanto, para encontrar el número de tuplas en la relación *asignatura*, se escribe:

select count (*)
from *asignatura*;

SQL no permite el uso de **distinct** con **count** (*). Es legal utilizar **distinct** con **max** y **min,** aunque el resultado no cambie. Se puede usar la palabra **all** en lugar de **distinct** para indicar que se mantengan los duplicados, pero como **all** es la opción por defecto, no hay necesidad de hacerlo.

3.7.2. Agregación con agrupamiento

A veces resulta útil aplicar una función de agregación no solo a un único conjunto de tuplas, sino también a un grupo de conjuntos de tuplas; esto se especifica en SQL usando la cláusula **group by.** El atributo o atributos indicados en la cláusula **group by** se usan para formar grupos. Las tuplas con el mismo valor en todos los atributos en la cláusula **group by** se sitúan en un grupo.

Como ejemplo, considere la siguiente consulta: «Encontrar el sueldo medio de cada departamento». Se escribe de la siguiente forma:

select *nombre_dept*, **avg** (*sueldo*) **as** *med_sueldo*
from *profesor*
group by *nombre_dept*;

En la Figura 3.14 se muestran las tuplas de la relación *profesor* agrupadas por el atributo *nombre_dept*, que es el primer paso en el cálculo del resultado de la consulta. La agregación especificada se calcula para cada uno de los grupos, y el resultado de esta consulta se muestra en la Figura 3.15.

Por otra parte, suponga la consulta: «Encontrar el sueldo medio de todos los profesores». Esta consulta se escribe de la siguiente forma:

select avg (*sueldo*)
from *profesor*;

En este caso se ha omitido la cláusula **group by,** por lo que toda la relación se trata como un único grupo.

Para ver otro ejemplo de agregación en grupos de tuplas, considere la consulta: «Encontrar el número de profesores de cada departamento que enseñó una asignatura en el semestre de primavera de 2010». La información sobre qué profesores enseñan qué secciones de asignaturas en qué semestre se encuentra en la relación *enseña*. Sin embargo, hay que unir esta información con la información de la relación *profesor* para obtener el nombre del departamento de cada profesor. Por tanto, la consulta se escribe como:

select *nombre_dept*, **count** (**distinct** *ID*) **as** *prof_cuenta*
from *profesor* **natural join** *enseña*
where *semestre* = ´Primavera´ **and** *año* = 2010
group by *nombre_dept*;

El resultado se muestra en la Figura 3.16.

Cuando una consulta de SQL utilice agrupamientos es importante asegurarse de que solo los atributos que aparecen en la sentencia **select** sin agregarse se encuentran en la cláusula **group by.** En otras palabras, cualquier atributo que no esté en la cláusula **group by** solo debe aparecer dentro de una función de agregación si aparece también en la cláusula **select;** en otro caso la consulta se trata como errónea. Por ejemplo, la siguiente consulta es errónea, ya que *ID* no aparece en la cláusula **group by** y sí aparece en la cláusula **select** sin haberse agregado:

/* consulta errónea */
select *nombre_dept*, *ID*, **avg** (*sueldo*)
from *profesor*
group by *nombre_dept*;

Un profesor de un determinado grupo (definido por su *nombre_dept*) puede tener un *ID* diferente, y como solo se obtiene una tupla como resultado de cada grupo, no existe una única forma de elegir qué valor de *ID* dar como salida. En consecuencia, estos casos no se permiten en SQL.

3.7.3. La cláusula «having»

Hay veces que es útil indicar una condición que se aplica a grupos y no a tuplas. Por ejemplo, podríamos estar interesados solo en aquellos departamentos en los que el sueldo medio de los profesores es superior a 42.000 €. Esta condición no se aplica a una sola tupla sino que se aplica a todos los grupos que se construyen con la cláusula **group by.** Para expresar esta consulta se utiliza la cláusula **having** de SQL. SQL aplica los predicados de la cláusula **having** después de haber hecho los agrupamientos, por lo que se pueden utilizar las funciones de agregación. Esta consulta se expresa en SQL como:

select *nombre_dept*, **avg** (*sueldo*) **as** *med_sueldo*
from *profesor*
group by *nombre_dept*
having avg (*sueldo*) > 42000;

El resultado se muestra en la Figura 3.17.

ID	*nombre*	*nombre_dept*	*sueldo*
76766	Crick	Biología	72000
45565	Katz	Informática	75000
10101	Srinivasan	Informática	65000
83821	Brandt	Informática	92000
98345	Kim	Electrónica	80000
12121	Wu	Finanzas	90000
76543	Singh	Finanzas	80000
32343	El Said	Historia	60000
58583	Califieri	Historia	62000
15151	Mozart	Música	40000
33456	Gold	Física	87000
22222	Einstein	Física	95000

Figura 3.14. Tuplas de la relación *profesor*, agrupadas por el atributo *nombre_dept*.

nombre_dept	*med_sueldo*
Biología	72000
Informática	77333
Electrónica	80000
Finanzas	85000
Historia	61000
Música	40000
Física	91000

Figura 3.15. Relación resultado de la consulta «Encontrar el sueldo medio en cada departamento».

nombre_dept	*prof_cuenta*
Informática	3
Finanzas	1
Historia	1
Música	1

Figura 3.16. Relación resultado de la consulta «Encontrar el número de profesores en cada departamento que enseñó alguna asignatura en el semestre de primavera de 2010».

nombre_dept	*avg(med_sueldo)*
Física	91000
Electrónica	80000
Finanzas	85000
Informática	77333
Biología	72000
Historia	61000

Figura 3.17. Relación resultado de la consulta «Encontrar el sueldo medio de los profesores en aquellos departamentos cuyo sueldo medio sea superior a 42.000 €».

Como en el caso de la cláusula **select,** cualquier atributo que se encuentre en la cláusula **having** sin haber estado agregado tiene que aparecer en la cláusula **group by,** en otro caso la consulta se trata como errónea.

El significado de una consulta que tenga cláusulas de agregación, **group by** o **having** se define de acuerdo a la siguiente secuencia de operaciones:

1. Como en el caso de las consultas sin agregación, la cláusula **from** se evalúa en primer lugar para obtener una relación.
2. Si existe una cláusula **where,** el predicado de la cláusula **where** se aplica al resultado de la relación de la cláusula **from.**
3. Las tuplas que satisfagan el predicado **where** se sitúan en grupos de acuerdo con la cláusula **group by,** si existe. Si la cláusula **group by** no existe, todo el conjunto de tuplas que satisfagan el predicado **where** se trata como si fuese un único grupo.
4. Se aplica la cláusula **having,** si existe, a cada uno de los grupos; los grupos que no satisfagan el predicado de la cláusula **having** se eliminan.
5. La cláusula **select** utiliza el resto de grupos para generar tuplas con el resultado de la consulta aplicando las funciones de agregación para obtener una única tupla resultado para cada grupo.

Para mostrar el uso tanto de la cláusula **having** como de la cláusula **where** en la misma consulta, suponga la siguiente consulta «Para cada sección de asignatura que se ofreció en 2009, encontrar el promedio de créditos totales (*tot_créditos)* de todos los estudiantes matriculados en la sección, si la sección tuvo al menos dos estudiantes»:

select *asignatura_id*, *semestre*, *año*, *secc_id*, **avg** (*tot_créditos*)
from *matricula* **natural join** *student*
where *año* = 2009
group by *asignatura_id*, *semestre*, *año*, *secc_id*
having count (*ID*) >= 2;

Fíjese en que toda la información necesaria para la consulta anterior se encuentra disponible en las relaciones *matricula* y *estudiante*, y aunque la consulta se refiere a secciones, no es necesaria la unión con *sección*.

3.7.4. Agregación con valores nulos y valores booleanos

Cuando existen valores nulos, el proceso de las operaciones de agregación se complica. Por ejemplo, suponga que algunas tuplas de la relación *profesor* tienen el valor *null* en *sueldo*. Considere la siguiente consulta para calcular el total de todos los sueldos:

select sum (*sueldo*)
from *profesor*;

Los valores que se deben sumar en la consulta anterior incluyen valores nulos, ya que algunas de las tuplas tienen un valor nulo en *sueldo*. En lugar de decir que la suma total vale *null*, SQL estándar indica que el operador **sum** debe ignorar todos los valores a *null* que aparezcan como entrada.

En general, las funciones de agregación tratan los nulos según la siguiente regla: todas las funciones de agregación excepto **count (*)** deben ignorar los valores nulos que aparezcan como entrada. Como resultado de esta regla de ignorar los valores nulos, la colección de valores de entrada puede estar vacía. Se define que **count** de una colección vacía debe valer 0, y el resto de operaciones de agregación devuelven un valor *null* cuando se aplican a una colección vacía. El efecto de los valores nulos en algunos de los constructores más complicados de SQL puede ser sutil.

Un tipo de datos **booleanos** puede tomar los valores **true, false** y **unknown,** como se incluyó en SQL:1999. Las funciones de agregación **some** (algunos) y **every** (todos), que significan lo que se puede suponer de forma intuitiva, se pueden aplicar solo a colecciones de valores booleanos.

3.8. Subconsultas anidadas

SQL proporciona un mecanismo para anidar subconsultas. Las subconsultas son expresiones **select-from-where** que están anidadas dentro de otra consulta. Una finalidad habitual de las subconsultas es llevar a cabo comprobaciones de pertenencia a conjuntos, hacer comparaciones de conjuntos y determinar cardinalidades de conjuntos.

Estos usos de las subconsultas anidadas se estudian en la cláusula **where** de las Secciones 3.8.1 a 3.8.4. En la Sección 3.8.5 se estudian las subconsultas anidadas en la cláusula **from.**

En la Sección 3.8.7 se ve cómo una clase de subconsultas denominadas subconsultas escalares pueden aparecer en cualquier parte de una expresión en la que se devuelva un valor.

3.8.1. Pertenencia a conjuntos

SQL permite comprobar la pertenencia de las tuplas a una relación. La conectiva **in** comprueba la pertenencia a un conjunto, donde el conjunto es la colección de valores resultado de una cláusula **select.** La conectiva **not in** comprueba la no pertenencia a un conjunto.

Como ejemplo, considérese de nuevo la consulta «Encontrar todas las asignaturas que se enseñaron en los semestres de otoño de 2009 y primavera de 2010». Anteriormente se escribió esta consulta como la intersección de dos conjuntos: el conjunto de las asignaturas que se enseñaron en el semestre de otoño de 2009 y el conjunto de asignaturas que se enseñaron en primavera de 2010. Se puede adoptar el enfoque alternativo, que consiste en determinar todas las asignaturas que se enseñaron en otoño de 2009 y son miembros del conjunto de asignaturas que se enseñaron en primavera de 2010. Claramente, esta formulación genera el mismo resultado que la anterior, pero obliga a formular la consulta usando la conectiva de SQL **in.** Se comienza determinando todas las asignaturas que se enseñaron en primavera de 2010 y para ello se escribe la consulta:

(**select** *asignatura_id*
from *sección*
where *semestre* = ´Primavera´ **and** *año* = 2010)

A continuación se determinan las asignaturas que se enseñaron en otoño de 2009 y que aparecen en el conjunto de asignaturas que aparecen en la subconsulta. Esto se consigue anidando la subconsulta en un **select** más externo. La consulta resultante es:

select distinct *asignatura_id*
from *sección*
where *semestre* = ´Otoño´ **and** *año* = 2009 **and**
asignatura_id **in** (**select** *asignatura_id*
from *sección*
where *semestre* = ´Primavera´
and *año* = 2010);

El ejemplo muestra que en SQL es posible escribir la misma consulta de diversas formas. Esta flexibilidad es de gran utilidad, puesto que permite a los usuarios pensar en las consultas del modo que les parezca más natural. Más adelante se verá que en SQL hay una gran cantidad de redundancia.

El constructor **not in** se usa de forma similar al constructor **in.** Por ejemplo, para encontrar todas las asignaturas que se enseñaron en el semestre de otoño de 2009, pero no en el semestre de primavera de 2010, se escribe:

```
select distinct asignatura_id
from sección
where semestre = ´Otoño´ and año = 2009 and
asignatura_id not in (select asignatura_id
                      from sección
                      where semestre = ´Primavera´
                            and año = 2010);
```

Los operadores **in** y **not in** también se pueden utilizar sobre conjuntos enumerados. La consulta siguiente selecciona los nombres de los profesores que no son ni «Mozart» ni «Einstein»:

```
select distinct nombre
from profesor
where nombre not in (´Mozart´, ´Einstein´);
```

En los ejemplos anteriores se comprueba la pertenencia en una relación usando un atributo. También es posible comprobar la pertenencia para cualquier relación arbitraria de SQL. Por ejemplo, se puede escribir la consulta «encontrar el número total de estudiantes (diferentes) que hayan cursado secciones de asignaturas en las que haya enseñado el profesor con *ID* 110011» de la siguiente forma:

```
select count (distinct ID)
from matricula
where (asignatura_id, secc_id, semestre, año)
  in (select asignatura_id, secc_id, semestre, año
      from enseña
      where enseña.ID = 10101);
```

3.8.2. Comparación de conjuntos

Como ejemplo de la capacidad de comparar conjuntos de las consultas anidadas, considérese la consulta «Encontrar los nombres de todos los profesores cuyo sueldo es mayor que al menos uno de los profesores del departamento de Biología». En la Sección 3.4.1 escribimos esta consulta de la siguiente forma:

```
select distinct T.nombre
from profesor as T, profesor as S
where T.sueldo > S.sueldo and S.nombre_dept = ´Biología´;
```

SQL ofrece, sin embargo, un estilo alternativo de formular la consulta anterior. La expresión: «mayor que al menos una» se representa en SQL por > **some.** Este constructor permite volver a escribir la consulta de forma más parecida a la formulación de la consulta en lenguaje natural:

```
select nombre
from profesor
where sueldo > some (select sueldo
                     from profesor
                     where nombre_dept = ´Biología´);
```

La subconsulta:

```
(select sueldo
 from profesor
 where nombre_dept = ´Biología´)
```

genera el conjunto de todos los valores de sueldo de todos los profesores del departamento de Biología. La comparación > **some** de la cláusula **where** de la cláusula **select** más externa es cierta si el valor de *sueldo* de la tupla es mayor que al menos un miembro del conjunto de todos los valores sueldo para los profesores de Biología.

SQL también permite realizar las comparaciones < **some,** <= **some,** >= **some,** = **some** y <> **some.** Como ejercicio, compruébese que = **some** es idéntico a **in,** mientras que <> **some** *no* es lo mismo que **not in.**[7]

Ahora se va a modificar ligeramente la consulta. Se trata de determinar los nombres de todos los profesores que tienen un valor de sueldo mayor al de cualquier profesor del departamento de Biología. El constructor > **all** se corresponde con la expresión «superior al de todos». Usando este constructor se puede escribir la consulta del modo siguiente:

```
select nombre
from profesor
where sueldo > all (select sueldo
                    from profesor
                    where nombre_dept = ´Biología´);
```

Al igual que con **some,** SQL también permite las comparaciones < **all,** <= **all,** >= **all,** = **all** y <> **all.** Como ejercicio, compruebe que <> **all** es lo mismo que **not in,** mientras que = **all** *no* es lo mismo que **in.**

Como ejemplo adicional de comparación de conjuntos, considere la consulta «Encontrar los departamentos que tienen los mayores promedios de sueldos». Escribiremos la función para calcular los sueldos medios y, después, la anidaremos como una subconsulta de una consulta mayor que encuentre aquellos departamentos para los que el sueldo promedio es mayor o igual que todos los promedios de sueldo:

```
select nombre_dept
from profesor
group by nombre_dept
having avg (sueldo) >= all (select avg (sueldo)
                            from profesor
                            group by nombre_dept);
```

3.8.3. Comprobación de relaciones vacías

SQL incluye la posibilidad de comprobar si las subconsultas tienen alguna tupla en su resultado. El constructor **exists** devuelve el valor **true** si su argumento subconsulta no resulta vacía. Mediante el constructor **exists** se puede formular la consulta «Encontrar todas las asignaturas que se enseñaron tanto en el semestre de otoño de 2009 como en el semestre de primavera de 2010» en otra forma distinta más:

```
select asignatura_id
from sección as S
where semestre = ´Otoño´ and año = 2009 and
      exists (select *
              from sección as T
              where semestre = ´Primavera´ and año = 2010
                and S.asignatura_id = T.asignatura_id);
```

La consulta anterior muestra una característica de SQL en la que el nombre de una correlación de una consulta externa (*S* en la consulta anterior) se puede utilizar en una subconsulta en la cláusula **where.** Una subconsulta que usa un nombre de correlación de una consulta externa se denomina **subconsulta de correlación.**

En las consultas que contienen subconsultas se aplica una regla de ámbito para los nombres de correlación. De acuerdo con

7 La palabra clave **any** es sinónimo de **some** en algunos SQL. Las primeras versiones de SQL solo permitían la palabra **any.** Las posteriores añadieron la alternativa **some** para evitar las ambigüedades lingüísticas de la palabra *any* (cualquiera) en lenguaje natural.

esta regla, en una subconsulta es legal utilizar solo los nombres de correlación definidos en la propia subconsulta o en cualquier consulta que la contenga. Si un nombre de correlación se define tanto localmente en la subconsulta, como globalmente en una consulta que la contenga, se aplica la definición local. Esta regla es análoga a las reglas de ámbito usuales para las variables en los lenguajes de programación.

Mediante el constructor **not exists** se puede comprobar la inexistencia de tuplas en el resultado de las subconsultas. Se puede utilizar el constructor **not exists** para simular la operación de continencia de conjuntos (es decir, superconjuntos). Se puede escribir «la relación *A* contiene a la relación *B*» como «**not exists** (B **except** A)» (aunque no forma parte de las normas SQL-92 ni SQL:1999, el operador **contains** aparecía en algunos de los primeros sistemas relacionales). Para ilustrar el operador **not exists,** considere la consulta «Encontrar a todos los estudiantes que se hayan matriculado en todas las asignaturas ofertadas». Mediante el constructor **except** se puede formular la consulta del modo siguiente:

```
select distinct S.ID, S.nombre
from estudiante as S
where not exists ((select asignatura_id
                   from asignatura
                   where nombre_dept = 'Biología')
                   except
                   (select T.asignatura_id
                   from matricula as T
                   where S.ID = T.ID));
```

de donde la subconsulta:

```
(select asignatura_id
 from asignatura
 where nombre_dept = 'Biología')
```

determina el conjunto de todas las asignaturas que se ofertan en el departamento de Biología. La subconsulta:

```
(select T.asignatura_id
 from matricula as T
 where S.ID = T.ID)
```

determina todas las asignaturas en las que se haya matriculado el estudiante *S.ID*. Por tanto, el **select** externo comprueba para cada estudiante si el conjunto de todas las asignaturas en las que se ha matriculado contiene al conjunto de todas las asignaturas que se ofertan en el departamento de Biología.

3.8.4. Comprobación de la ausencia de tuplas duplicadas

SQL incluye la posibilidad de comprobar si las subconsultas tienen tuplas duplicadas en su resultado. El constructor[8] **unique** devuelve el valor **true** si la subconsulta que se le pasa como argumento no contiene tuplas duplicadas. Mediante el constructor **unique** se puede formular la consulta «Encontrar todas las asignaturas que se ofertaron al menos una vez en 2009» del modo siguiente:

```
select T.asignatura_id
from asignatura as T
where unique (select R.asignatura_id
              from sección as R
              where T.asignatura_id = R.asignatura_id and
                    R.año = 2009);
```

8 Este constructor aún no se encuentra muy implantado.

Fíjese en que si una asignatura no se ofrece en 2009, la subconsulta debería devolver un resultado vacío, y el predicado **unique** debería evaluarse a cierto para el conjunto vacío.

Una versión equivalente de la consulta anterior que no utiliza el constructor **unique** es:

```
select T.asignatura_id
from asignatura as T
where 1 <= (select count(R.asignatura_id)
            from sección as R
            where T.asignatura_id = R.asignatura_id and
                  R.año = 2009);
```

La existencia de tuplas duplicadas en una subconsulta se puede comprobar mediante el constructor **not unique.**

Para ilustrar este constructor, considérese la consulta «Encontrar todas las asignaturas que se han ofrecido al menos dos veces en 2009».

Se puede formular de la siguiente manera:

```
select T.asignatura_id
from asignatura as T
where not unique (select R.asignatura_id
                  from sección as R
                  where T.asignatura_id = R.asignatura_id
                    and R.año = 2009);
```

Formalmente, la evaluación de **unique** sobre una relación se define como falsa si y solo si la relación contiene dos tuplas t_1 y t_2 tales que $t_1 = t_2$. Como la comprobación $t_1 = t_2$ es falsa, si algún campo de t_1 o de t_2 es nulo, es posible que el resultado de **unique** sea cierto aunque haya varias copias de una misma tupla, siempre que al menos uno de los atributos de la tupla sea nulo.

3.8.5. Subconsultas en la cláusula from

SQL permite que se incluya una subconsulta en la cláusula **from.** El concepto clave es que cualquier expresión **select-from-where** devuelve una relación como resultado y, por tanto, se puede insertar en otra **select-from-where** en cualquier parte en la que pueda aparecer una relación.

Suponga la consulta «Encontrar los salarios promedio de los profesores de aquellos departamentos en los que el sueldo medio es superior a 42.000 €». Ya hemos escrito esta consulta en la Sección 3.7 utilizando la cláusula **having.** Ahora se puede escribir sin utilizar la cláusula **having,** utilizando una subconsulta en la cláusula **from.**

Lo podemos hacer de la siguiente forma:

```
select nombre_dept, med_sueldo
from (select nombre_dept, avg (sueldo) as med_sueldo
      from profesor
      group by nombre_dept)
where med_sueldo > 42000;
```

La subconsulta genera una relación que consiste en los nombres de todos los departamentos y los correspondientes sueldos medios de los profesores. Los atributos de la subconsulta resultado se pueden utilizar en una consulta externa, como se puede ver en el ejemplo anterior.

Está claro que no se necesita la cláusula **having,** ya que la subconsulta de la cláusula **from** calcula el sueldo medio y el predicado que se usaba en la cláusula **having** anterior ahora está en la cláusula **where** de la consulta externa.

Se puede dar a la relación resultado de la subconsulta un nombre, y renombrar los atributos, utilizando la cláusula **as** como se indica:

```
select nombre_dept, sueldo_medio
from (select nombre_dept, avg (sueldo)
        from profesor
        group by nombre_dept)
        as dept_ medio (nombre_dept, sueldo_medio)
where sueldo_medio > 42000;
```

La relación resultado de la subconsulta se ha llamado *dept_medio*, con los atributos *nombre_dept* y *sueldo_medio*.

La mayoría, pero no todas, las implementaciones de SQL permiten las subconsultas anidadas de la cláusula **from.** Sin embargo, algunas implementaciones de SQL, especialmente Oracle, no permiten el renombrado de la relación resultado de una cláusula **from.**

Otro ejemplo adicional: suponga que se desea encontrar el máximo de todos los departamentos del sueldo total de cada departamento. La cláusula **having** no es de ayuda para esta tarea, pero se puede escribir esta consulta fácilmente utilizando una subconsulta en la cláusula **from,** como:

```
select max (tot_sueldo)
from (select nombre_dept, sum(sueldo)
        from profesor
        group by nombre_dept)
          as dept_total (nombre_dept, tot_sueldo);
```

Hay que indicar que las subconsultas anidadas en la cláusula **from** no pueden utilizar variables de correlación de otras relaciones en la cláusula **from.** Sin embargo, SQL:2003 permite que una subconsulta en la cláusula **from** utilice un prefijo con la palabra clave **lateral** para acceder a los atributos de las tablas precedentes o a subconsultas en la cláusula **from.** Por ejemplo, si se quiere imprimir los nombres de todos los profesores junto con su sueldo y el sueldo medio de su departamento, se podría escribir:

```
select nombre, sueldo, sueldo_medio
from profesor I1, lateral (select avg(sueldo) as sueldo_medio
                           from profesor I2
                           where I2.nombre_dept =
                                 I1.nombre_dept);
```

Sin utilizar la cláusula **lateral,** la subconsulta no puede acceder a la variable de correlación *I1* desde la consulta externa. Actualmente solo algunas pocas implementaciones de SQL, como DB2 de IBM, admiten la cláusula **lateral.**

3.8.6. La cláusula with

La cláusula **with** proporciona una forma de definir vistas temporales cuya definición solo está disponible para la consulta en la que aparece la cláusula. Considérese la siguiente consulta, que selecciona aquellos departamentos con el máximo presupuesto:

```
with max_presupuesto (valor) as
      (select max(presupuesto)
       from departamento)
select presupuesto
from departamento, max_presupuesto
where departamento.presupuesto = max_presupuesto.valor;
```

La cláusula **with** define la relación temporal *max_presupuesto*, que se usa en la consulta inmediatamente siguiente. La cláusula **with,** introducida en SQL:1999, es admitida por casi todos, pero no todos, los sistemas de bases de datos.

La consulta anterior se podría haber escrito usando una subconsulta anidada, bien en la cláusula **from,** bien en la **where.** Sin embargo, el uso de subconsultas anidadas hace que la consulta sea más difícil de leer y de entender. La cláusula **with** hace que la lógica de la consulta sea más clara; también permite utilizar definiciones de vistas en varias partes de la consulta.

Por ejemplo, suponga que se desean encontrar los departamentos en los que el sueldo total es mayor que el sueldo medio de todos los sueldos de todos los departamentos. Se puede escribir la consulta usando la cláusula **with** como:

```
with dept_total (nombre_dept, valor) as
      (select nombre_dept, sum(sueldo)
      from profesor
      group by nombre_dept),
dept_total_medio(valor) as
      (select avg(valor)
      from dept_total)
select nombre_dept
from dept_total, dept_total_medio
where dept_total.valor >= dept_total_medio.valor;
```

Por supuesto, se puede crear una consulta equivalente sin utilizar la cláusula **with,** pero sería mucho más compleja y difícil de entender. Puede escribirla a modo de ejercicio.

3.8.7. Subconsultas escalares

En SQL se permite escribir subconsultas en cualquier punto en el que una expresión devuelve un valor; estas subconsultas se llaman **subconsultas escalares.** Por ejemplo, se puede usar una subconsulta en la cláusula **select** como se muestra en el siguiente ejemplo que lista todos los departamentos junto con el número de profesores de cada departamento.

```
select nombre_dept,
        (select count(*)
        from profesor
        where departamento.nombre_dept =
              profesor.nombre_dept)
        as num_profesores
from departamento;
```

La subconsulta del ejemplo anterior se garantiza que devuelve un único valor, ya que tiene un agregado **count(*)** sin **group by.** El ejemplo también muestra el uso de la variable de correlación; es decir, atributos de relaciones en la cláusula **from** de la consulta externa, como *departamento.nombre_dept* en el ejemplo anterior.

Las subconsultas escalares pueden darse en las cláusulas **select, where** y **having.** Las subconsultas escalares también se pueden definir sin agregación. No siempre es posible identificar en tiempo de compilación si una subconsulta puede devolver más de un tupla como resultado; si el resultado tiene más de una tupla cuando se ejecuta la subconsulta, ocurrirá un error en tiempo de ejecución.

Fíjese en que técnicamente el tipo del resultado de una subconsulta escalar sigue siendo una relación, incluso aunque contenga solo una tupla. Sin embargo, cuando se usa una subconsulta escalar en una expresión en la que se espera un valor, SQL extrae implícitamente el valor de dicho atributo único de la tupla única de la relación, y devuelve dicho valor.

3.9. Modificación de la base de datos

Hasta ahora se ha estudiado la extracción de información de las bases de datos. A continuación se mostrará cómo añadir, eliminar y modificar información utilizando SQL.

3.9.1. Borrado

Las solicitudes de borrado se expresan casi igual que las consultas. Solo se pueden borrar tuplas completas; no se pueden borrar únicamente valores de atributos concretos. SQL expresa los borrados mediante:

delete from *r*
where *P*;

donde *P* representa un predicado y *r* representa una relación. La declaración **delete** busca primero todas las tuplas *t* en *r* para las que *P* (*t*) es cierto y a continuación las borra de *r*. La cláusula **where** se puede omitir, en cuyo caso se borran todas las tuplas de *r*.

Obsérvese que cada comando **delete** solo opera sobre una relación. Si se desea borrar tuplas de varias relaciones hay que utilizar una orden **delete** por cada relación. El predicado de la cláusula **where** puede ser tan complicado como la cláusula **where** de cualquier orden **select.** En el otro extremo, la cláusula **where** puede estar vacía. La consulta:

delete from *profesor*;

borra todas las tuplas de la relación *profesor*. La relación *profesor* sigue existiendo, pero está vacía.

A continuación se muestran una serie de ejemplos de peticiones de borrado en SQL:

- Borrar todas las tuplas de la relación *profesor* que pertenezcan al departamento Finanzas:

  delete from *profesor*
  where *nombre_dept* = ´Finanzas´;

- Borrar todos los profesores con un sueldo entre 13.000 € y 15.000 €:

  delete from *profesor*
  where *sueldo* **between** 13000 **and** 15000;

- Borrar todas las tuplas en la relación *profesor* para las cuales los profesores estén asociados con un departamento que se encuentra en el edificio Watson:

  delete from *profesor*
  where *nombre_dept* **in** (**select** *nombre_dept*
  from *departamento*
  where *edificio* = ´Watson´);

  La consulta **delete** primero determina todos los departamentos ubicados en el edificio Watson y después borra todas las tuplas *profesor* que pertenecen a dichos departamentos.

Observe que, aunque solo se pueden borrar tuplas de una sola relación a la vez, se puede referenciar cualquier número de relaciones en un **select-from-where** anidado en la cláusula **where** de un **delete.** La solicitud **delete** puede contener un **select** anidado con referencias a la relación de la que borrar las tuplas. Por ejemplo, suponga que se desea borrar los registros de todos los profesores con un sueldo inferior a la media de la universidad. Se puede escribir:

delete from *profesor*
where *sueldo* < (**select avg** (*sueldo*)
from *profesor*);

La sentencia **delete** primero comprueba cada tupla de la relación *profesor* para verificar si el sueldo es menor que el sueldo medio de los profesores de la universidad. Después, todas las tuplas que fallan en la comprobación; es decir, representan a un profesor con un sueldo por debajo de la media, se borran. Es importante realizar las comprobaciones antes de los borrados, si hay tuplas que se borran antes que otras se hayan comprobado, el sueldo medio puede cambiar y el resultado final de **delete** dependería del orden en que se procesasen las tuplas.

3.9.2. Inserción

Para insertar datos en una relación se especifica la tupla que se desea insertar o se formula una consulta cuyo resultado sea el conjunto de tuplas que se desea insertar. Obviamente, los valores de los atributos de las tuplas que se inserten deben pertenecer al dominio de los atributos. De igual modo, las tuplas insertadas deben tener el número correcto de atributos.

La instrucción **insert** más sencilla es una solicitud de inserción de una tupla. Supóngase que se desea insertar el hecho de que hay una asignatura CS-437 en el departamento de Informática con el nombre «Bases de datos», de 4 créditos. Se escribe:

insert into *asignatura*
values (´CS-437´, ´Bases de datos´, ´Informática´, 4);

En este ejemplo los valores se especifican en el mismo orden en que aparecen los atributos correspondientes en el esquema de la relación. Para beneficio de los usuarios, que puede que no recuerden el orden de los atributos, SQL permite que los atributos se especifiquen en la cláusula **insert.** Por ejemplo, las siguientes instrucciones **insert** tienen una función idéntica a la anterior:

insert into *asignatura* (*asignatura_id*, *nombre*,
nombre_dept, *créditos*)
values (´CS-437´, ´Bases de datos´, ´Informática´, 4);

insert into *asignatura* (*nombre*, *asignatura_id*, *créditos*,
nombre_dept)
values (´Bases de datos´, ´CS-437´, 4, ´Informática´);

De forma más general es posible que se desee insertar las tuplas que resultan de una consulta. Supóngase que queremos convertir en profesores del departamento de Música, con un sueldo de 18.000 €, a todos los estudiantes del departamento de Música que hayan conseguido más de 144 créditos. Se escribirá:

insert into *profesor*
select *ID*, *nombre*, *nombre_dept*, 18000
from *estudiante*
where *nombre_dept* = ´Música´ **and** *tot_cred* > 144;

En lugar de especificar una tupla, como se hizo en los primeros ejemplos de este apartado, se utiliza una instrucción **select** para especificar un conjunto de tuplas. SQL evalúa en primer lugar la instrucción **select,** lo que produce un conjunto de tuplas que se inserta a continuación en la relación *profesor*. Cada tupla tiene un *ID*, un *nombre*, un *nombre_dept* (Música) y un sueldo de 18.000 €.

Es importante que se evalúe la sentencia **select** completamente antes de llevar a cabo cualquier inserción. Cualquier inserción que se produzca incluso cuando la sentencia **select** se está evaluando, como por ejemplo en:

insert into *estudiante*
select *
from *estudiante*;

podría insertar un número infinito de tuplas, si no existe restricción de clave primaria en *estudiante*. Sin clave primaria, la solicitud insertaría la primera tupla de nuevo en *estudiante*, creando así una segunda copia de la tupla. Como esta segunda copia ya forma parte de *estudiante*, la instrucción **select** puede encontrarla, y se insertaría una tercera copia en *estudiante*. La instrucción **select** puede entonces encontrar esta tercera copia e insertar una cuarta copia, y así indefinidamente. Al evaluar por completo la instrucción **select** antes de realizar ninguna inserción se evita este tipo de problemas. Por tanto, la sentencia **insert** anterior simplemente duplicaría todas las tuplas de la relación *estudiante*, si la relación no tiene restricciones de clave primaria.

La discusión de la instrucción **insert** solo ha considerado ejemplos en los que se especificaba un valor para cada atributo de las tuplas insertadas. Es posible dar valores únicamente a algunos de los atributos del esquema para las tuplas insertadas. A los atributos restantes se les asigna un valor nulo, que se denota por *null*. Considere la consulta:

```
insert into estudiante
        values ('3003', 'Green', 'Finanzas', null);
```

La tupla que inserta esta petición especifica un estudiante con *ID* «3003» que es del departamento Finanzas, pero no se sabe el valor de *tot_créditos* del estudiante. Suponga la siguiente consulta:

```
select esudiante
from estudiante
where tot_créditos > 45;
```

Como no se conoce el valor de *tot_créditos* del estudiante «3003», no se puede determinar si es mayor que 45 o no.

La mayor parte de los productos de bases de datos tienen utilidades especiales de «carga masiva» para insertar en las relaciones grandes conjuntos de tuplas. Estas utilidades permiten leer datos de archivos de texto con formato y pueden ejecutarse mucho más rápido que la secuencia equivalente de instrucciones **insert.**

3.9.3. Actualizaciones

En determinadas situaciones puede ser deseable modificar un valor dentro de una tupla sin cambiar *todos* los valores de la misma. Para este tipo de situaciones se puede utilizar la instrucción **update.** Al igual que ocurre con **insert** y **delete,** se pueden elegir las tuplas que se van a actualizar mediante una consulta.

Suponga que se va a incrementar el sueldo anual, de forma que todos los sueldos de los profesores se incrementarán en un 5 por ciento. Escribimos:

```
update profesor
set sueldo = sueldo * 1.05;
```

Esta instrucción de actualización se aplica una vez a cada tupla de la relación *profesor*.

Si el incremento de sueldo solo se va a realizar a los profesores con un sueldo de menos de 70.000 €, escribimos:

```
update profesor
set sueldo = sueldo * 1.05
where sueldo < 70000;
```

En general, la cláusula **where** de la instrucción **update** puede contener cualquier constructor permitido en la cláusula **where** de la instrucción **select** (incluyendo instrucciones **select** anidadas). Al igual que ocurre con **insert** y con **delete,** un **select** anidado en una instrucción **update** puede hacer referencia a la relación que se esté actualizando. Al igual que antes, SQL primero comprueba todas las tuplas de la relación para determinar si se deben actualizar y después realiza la actualización. Por ejemplo, se puede escribir la solicitud «Dar un incremento del 5 por ciento a los profesores cuyo sueldo sea inferior a la media» como sigue:

```
update profesor
set sueldo = sueldo * 1.05
where sueldo < (select avg (sueldo)
                from profesor);
```

Supóngase que todos los profesores con sueldos superiores a 100.000 € reciben un 3 por ciento de aumento, mientras que el resto recibe un 5 por ciento de incremento. Se pueden escribir dos instrucciones **update:**

```
update profesor
set sueldo = sueldo * 1.03
where sueldo > 100000;

update profesor
set sueldo = sueldo * 1.05
where sueldo <= 100000;
```

Obsérvese que el orden de las dos instrucciones **update** es importante. Si se cambiara el orden de las dos instrucciones, un profesor con un sueldo justo por debajo de los 100.000 € recibiría un incremento de sueldo del 8 por ciento.

SQL ofrece un constructor **case** que se puede utilizar para llevar a cabo las dos instrucciones de actualización anteriores en una única instrucción **update,** evitando el problema del orden de actualización.

```
update profesor
set sueldo = case
                when sueldo <= 100000 then sueldo * 1.05
                else sueldo * 1.03
            end
```

La forma general de la instrucción **case** es la siguiente:

```
case
    when pred_1 then resultado_1
    when pred_2 then resultado_2
    ...
    when pred_n then resultado_n
    else resultado_0
end
```

La operación devuelve $resultado_i$, donde i es el primero de $pred_1$, $pred_2$, …, $pred_n$ que se satisface; si ninguno de ellos se satisface, la operación devuelve $resultado_0$. Las instrucciones **case** se pueden usar en cualquier lugar en el que se espere un valor.

Las subconsultas escalares también son útiles en sentencias de actualización de SQL, donde se pueden utilizar en las cláusulas **set.** Suponga una actualización en la que se establece el atributo *tot_créditos* de cada tupla *estudiante* con la suma de los créditos de las asignaturas que hayan aprobado ya. Supongamos que una asignatura se da por aprobada si el estudiante tiene una calificación que no sea ni 'F' ni *null*. Para especificar esta actualización se necesita utilizar una subconsulta en la cláusula **set,** como se muestra a continuación:

```
update estudiante S
set tot_créditos = (
    select sum(créditos)
    from matricula natural join asignatura
    where S.ID = matricula.ID and
                 matricula.nota <> 'F' and
                 matricula.nota is not null);
```

Observe que la subconsulta usa una variable de correlación *S* en la sentencia de actualización. Si un estudiante no ha aprobado ninguna asignatura, la sentencia de actualización debería poner el valor del atributo *tot_créditos* a *null*. Para establecer a 0 este valor en lugar de a *null*, podríamos utilizar otra sentencia **update** para sustituir los valores *null* por valores 0; una alternativa mejor es sustituir la cláusula «**select sum**(*créditos*)» en la subconsulta precedente por la siguiente cláusula **select** usando una expresión **case:**

```
select case
    when sum(créditos) is not null then sum(créditos)
    else 0
    end
```

3.10. Resumen

- SQL es el lenguaje de consulta relacional más influyente comercialmente. Consta de varias partes:
  - **Lenguaje de definición de datos** (LDD), que proporciona comandos para definir esquemas de relación, borrar relaciones y modificar los esquemas.
  - **Lenguaje de manipulación de datos** (LMD), que incluye el lenguaje de consultas y comandos para insertar, borrar y modificar tuplas de la base de datos.
- El lenguaje de definición de datos de SQL se utiliza para crear relaciones con esquemas especificados. Además de especificar los nombres y los tipos de los atributos de las relaciones, también permite la especificación de restricciones de integridad, como las restricciones de clave primaria y las de clave externa.
- SQL incluye varios constructores del lenguaje de consultas a la base de datos. Entre ellos incluye las cláusulas **select, from** y **where,** así como soporte para la operación unión natural.
- SQL también proporciona mecanismos para el renombrado de atributos y relaciones y para ordenar los resultados según determinados atributos.
- SQL admite las operaciones básicas de conjuntos sobre relaciones como **union, intersect** y **except,** que se corresponden con las operaciones matemáticas de teoría de conjuntos $\cup$, $\cap$ y $-$.
- SQL maneja consultas sobre relaciones que contengan valores nulos añadiendo el valor de verdad «*unknown*» a los valores usuales de verdad de cierto (*true*) y falso (*false*).
- SQL dispone de agregación, incluyendo la posibilidad de dividir relaciones en grupos y aplicar agregaciones de forma separada para cada grupo, así como de operaciones de conjuntos sobre los grupos.
- SQL dispone de subconsultas anidadas en las cláusulas **where** y **from** de una consulta exterior. También admite subconsultas escalares, en cualquier punto en el que se permita devolver un valor.
- SQL proporciona construcciones para la actualización, inserción y borrado de información.

Términos de repaso

- LDD: lenguaje de definición de datos.
- LMD: lenguaje de manipulación de datos.
- Esquema de la base de datos.
- Ejemplar de la base de datos.
- Esquema de relación.
- Ejemplar de relación.
- Clave primaria.
- Clave externa.
  - Relación referente.
  - Relación referenciada.
- Valor nulo (*null*).
- Lenguaje de consulta.
- Estructura de consultas de SQL.
  - Cláusula **select.**
  - Cláusula **from.**
  - Cláusula **where.**
- Operación de unión natural.
- Cláusula **as.**
- Cláusula **order by.**
- Nombre de correlación (variable de correlación, variable de tupla).
- Operaciones de conjuntos.
  - **union.**
  - **intersect.**
  - **except.**
- Valores nulos.
  - Valor de verdad «*unknown*».
- Funciones de agregación.
  - **avg, min, max, sum, count.**
  - **group by.**
  - **having.**
- Subconsultas anidadas.
- Comparación de conjuntos.
  - {<, <=, >, >=}{**some, all**}.
  - **exists.**
  - **unique.**
- Cláusula **lateral.**
- Cláusula **with.**
- Subconsulta escalar.
- Modificación de la base de datos.
  - Borrado.
  - Inserción.
  - Actualización.

Ejercicios prácticos

3.1. Escriba las siguientes consultas en SQL, utilizando el esquema de universalidad (se sugiere que ejecuten realmente estas consultas en una base de datos, utilizando los datos de muestra que proporcionamos en el sitio web del libro dbbook.com. En el mismo sitio web puede encontrar instrucciones para configurar la base de datos y cargar los datos de ejemplo).

a. Determine los nombres de las asignaturas del departamento de Informática que tengan 3 créditos.

b. Determine los ID de todos los estudiantes a quienes les enseñó un profesor de nombre Einstein; asegúrese de que no existan duplicados en el resultado.

c. Determine el mayor de los sueldos de todos los profesores.

d. Determine todos los profesores que ganan el mayor sueldo (puede que haya más de uno con el mismo sueldo).

e. Determine las matrículas de cada sección que se ofertaron en el otoño de 2009.

f. Determine la matrícula máxima, de entre todas las secciones, en el otoño de 2009.

g. Determine las secciones que tuvieron la matriculación máxima en el otoño de 2009.

3.2. Suponga que se tiene la relación *nota_puntos* (*nota*, *puntos*), para realizar la conversión de notas en letras a notas en valor numérico para la relación *matricula;* por ejemplo, una nota de «A» se podría traducir para que se corresponda con una nota de 4 puntos, una «A−» a 3.7 puntos, una «B^+» a 3.3 puntos, una «B» a 3 puntos, y así sucesivamente. Los

puntos que consigue un estudiante en una asignatura de las ofertadas (sección) se definen como el número de créditos de la asignatura multiplicado por el valor numérico de los puntos para la nota que el estudiante ha conseguido.

Teniendo en cuenta la relación anterior y nuestro esquema de universidad, escriba las siguientes consultas en SQL. Para simplificar, suponga que no existen tuplas en *matrícula* con valores nulos para *nota*.

a. Determine el número total de puntos que ha conseguido el estudiante con ID 12345, en todas las asignaturas en las que se ha matriculado.

b. Determine el promedio de puntos (*grade-point average*: GPA) del estudiante anterior, es decir, el número total de puntos dividido por el total de créditos de las asignaturas correspondientes.

c. Determine el ID y el promedio de puntos de todos los estudiantes.

3.3. Escriba las siguientes inserciones, borrados y actualizaciones en SQL utilizando el esquema de la universidad.

a. Aumentar el sueldo de todos los profesores del departamento de Informática en un 10 por ciento.

b. Borrar todas las asignaturas que nunca se hayan ofertado, es decir, que no existan en la relación *sección*.

c. Insertar a todos los estudiantes cuyo número de *tot_créditos* sea mayor de 100 como si fueran profesores en el mismo departamento con un sueldo de 10.000 €.

3.4. Considere la base de datos de la Figura 3.18, en la que las claves primarias están subrayadas. Escriba las siguientes consultas SQL para esta base de datos relacional.

a. Determine el número total de personas propietarias de un coche que estuvieron implicadas en un accidente en 2009.

b. Añada un nuevo accidente a la base de datos; invéntese los valores que necesite para los atributos.

c. Elimine el Mazda que pertenece a «Juan Gómez».

3.5. Suponga que se tiene la relación *notas*(ID, *puntos*) y se desea asignar notas con letras a los estudiantes según sus puntos de la siguiente forma: nota vale F si $puntos < 40$, nota vale C si $40 \leq puntos < 60$, nota vale B si $60 \leq puntos < 80$, y nota vale A si $80 \leq puntos$. Escriba las siguientes consultas en SQL:

a. Muestre la nota con letra de todos los estudiantes según esta relación *notas*.

b. Determine el número de estudiantes que ha conseguido cada una de las distintas notas con letra.

3.6. El operador de SQL **like** es sensible a las mayúsculas, pero se puede usar la función lower() sobre cadenas de caracteres para realizar una comparación sin tener en cuenta las mayúsculas. Para ver cómo hacerlo, escriba una consulta que determine los departamentos cuyos nombres contienen la cadena «mat» como subcadena, independientemente de las mayúsculas.

3.7. Considere la consulta en SQL:

select distinct $p.a1$
from p, $r1$, $r2$
where $p.a1 = r1.a1$ **or** $p.a1 = r2.a1$

¿En qué condiciones la consulta anterior selecciona valores de $p.a1$ que están en $r1$ o en $r2$? Examine cuidadosamente los casos en que alguno de los $r1$ o $r2$ pueda estar vacío.

3.8. Considere la base de datos de un banco de la Figura 3.19, en la que se han subrayado las claves primarias. Construya las siguientes consultas de SQL para esta base de datos relacional.

a. Determine todos los clientes del banco que tengan una cuenta pero no un préstamo.

b. Determine los nombres de todos los clientes que viven en la misma calle y ciudad que «Juan Gómez».

c. Determine los nombres de todas las sucursales con clientes que tengan una cuenta en el banco y vivan en «Miami».

3.9. Considere la base de datos de empleados de la Figura 3.20, en la que se han subrayado las claves primarias. Escriba una expresión en SQL para las siguientes consultas.

a. Determine los nombres y las ciudades de residencia de todos los empleados que trabajan para la «First Bank Corporation».

b. Determine los nombres, la calle y la ciudad de residencia de todos los empleados que trabajan para la «First Bank Corporation» y ganan más de 10.000 €.

c. Determine todos los empleados de la base de datos que no trabajan para la «First Bank Corporation».

d. Determine todos los empleados de la base de datos que ganan más que cualquier empleado de «Small Bank Corporation».

e. Suponga que las empresas pueden estar ubicadas en varias ciudades. Determine todas las compañías ubicadas en todas las ciudades en las que también está «Small Bank Corporation».

f. Determine la empresa que tiene mayor número de empleados.

g. Determine aquellos empleados que ganan, como media, mayor sueldo que el sueldo medio de los empleados de «First Bank Corporation».

3.10. Considere la siguiente base de datos relacional de la Figura 3.20. Escriba una expresión en SQL para las siguientes consultas:

a. Modifique la base de datos para que «Jones» viva en «Newtown».

b. Conceda a todos los gestores de «First Bank Corporation» un 10 por ciento de subida de sueldo, a no ser que el sueldo sea mayor de 100.000 €; en cuyo caso solo se les incrementa un 3 por ciento.

conductor (número_carné, nombre, dirección)
coche (matrícula, modelo, año)
accidente (número_parte, fecha, lugar)
posee (número_carné, matrícula)
participó (número_parte, matrícula, número_carné, importe_daños)

Figura 3.18. Base de datos de seguros para los Ejercicios 3.4 y 3.14.

sucursal(sucursal_nombre, sucursal_ciudad, activos)
cliente (cliente_nombre, cliente_calle, cliente ciudad)
préstamo (préstamo_número, sucursal_nombre, cantidad)
prestamista (cliente_nombre, préstamo_número)
cuenta (cuenta_ número, sucursal_nombre, saldo)
depositante (cliente_nombre, cuenta_número)

Figura 3.19. Base de datos de un banco para los Ejercicios 3.8 y 3.15.

empleado (empleado_nombre, calle, ciudad)
trabaja (empleado_nombre, empresa_nombre, sueldo)
empresa (empresa_nombre, ciudad)
gestiona (empleado_nombre, gestor_nombre)

Figura 3.20. Base de datos de empleados para los Ejercicios 3.9, 3.10, 3.16, 3.17 y 3.20.

Ejercicios

3.11. Escriba las siguientes consultas en SQL usando el esquema de la universidad.

a. Determine los nombres de todos los estudiantes que se hayan matriculado en alguna asignatura del departamento de Informática; asegúrese de que no existen nombres duplicados en el resultado.

b. Determine los ID y los nombres de todos los estudiantes que no se hayan matriculado en ninguna asignatura de las ofertadas en el semestre de primavera de 2009.

c. Para todos los departamentos, determine el mayor sueldo de los profesores de dicho departamento. Se puede suponer que todos los departamentos tienen al menos un profesor.

d. Determine el menor, de todos los departamentos, de todos los mayores sueldos calculados en la consulta anterior.

3.12. Escriba las siguientes consultas en SQL usando el esquema de la universidad.

a. Cree una nueva asignatura «CS-001», de nombre «Seminario semanal», con 0 créditos.

b. Cree una sección de esta asignatura en el semestre de otoño de 2009, con *secc_id* de 1.

c. Matricule a todos los estudiantes del departamento de Informática en la sección anterior.

d. Borre de todos los matriculados anteriormente a los estudiantes cuyo nombre sea Chávez.

e. Borre la asignatura CS-001. ¿Qué ocurre si se ejecuta esta sentencia de borrado sin haber borrado previamente las ofertas (secciones) de esta asignatura?

f. Borre todas las tuplas *matricula* correspondientes a todas las secciones de cualquier asignatura con la palabra «base» como parte del nombre de la asignatura; ignore la diferencia de mayúsculas cuando compare los nombres.

3.13. Escriba el LDD de SQL correspondiente al esquema de la Figura 3.18. Realice suposiciones razonables sobre los tipos, y asegúrese de declarar las claves primarias y externas.

3.14. Considere la base de datos de seguros de la Figura 3.18, en la que las claves primarias están subrayadas. Escriba las siguientes consultas de SQL para esta base de datos relacional.

a. Determine el número de accidentes en los que están involucrados coches que pertenecen a «Juan Gómez».

b. Actualice el importe de los daños del coche con matrícula «AABB2000» en el accidente con número «AR2197» a 3.000 €.

3.15. Considere la base de datos del banco de la Figura 3.19, en la que las claves primarias están subrayadas. Escriba estas consultas de SQL para esta base de datos relacional.

a. Determine todos los clientes que tengan una cuenta en *todas* las sucursales de «Brooklyn».

b. Determine la suma total de todos los préstamos del banco.

c. Determine los nombres de todas las sucursales que tengan activos superiores a al menos alguna de las sucursales situadas en «Brooklyn».

3.16. Considere la base de datos de empleados de la Figura 3.20, en la que las claves primarias están subrayadas. Escriba estas consultas de SQL para esta base de datos relacional.

a. Determine los nombres de todos los empleados que trabajan para la «First Bank Corporation».

b. Determine todos los empleados de la base de datos que viven en las mismas ciudades que las empresas para las que trabajan.

c. Determine los empleados de la base de datos que viven en la misma ciudad y calle que sus gestores.

d. Determine todos los empleados que ganan más que la media de los sueldos de todos los empleados de la empresa.

e. Determine la empresa que ha tenido el menor coste de sueldos.

3.17. Considere la base de datos de empleados de la Figura 3.20. Escriba estas consultas de SQL para esta base de datos relacional.

a. Conceda a todos los empleados de «First Bank Corporation» un 10 por ciento de incremento de sueldo.

b. Conceda a todos los gestores de «First Bank Corporation» un 10 por ciento de incremento de sueldo.

c. Borre todas las tuplas de la relación trabaja para los empleados de «Small Bank Corporation».

3.18. Indique dos razones por las que haya que introducir valores nulos en una base de datos.

3.19. Demuestre que, en SQL, <> **all** es idéntico que **not in.**

3.20. Escriba una definición de esquema de SQL para la base de datos de empleados de la Figura 3.20. Elija un dominio apropiado para cada uno de los atributos y las claves primarias apropiadas para cada esquema de relación.

3.21. Considere la base de datos de una biblioteca de la Figura 3.21. Escriba las siguientes consultas en SQL.

a. Escriba los nombre de los miembros que hayan tenido prestado cualquier libro publicado por «McGraw-Hill».

b. Escriba los nombres de los miembros que hayan tenido prestados todos los libros editados por «McGraw-Hill».

c. Para cada editorial, escriba los nombres de los miembros que hayan tenido prestados más de cinco libros de dicha editorial.

d. Escriba el número medio de libros que han tenido prestados los miembros. Considere que si un miembro no ha tenido prestado ningún libro, ese miembro no aparece en la relación *prestado*.

3.22. Reescriba la cláusula **where:**

where unique (**select** *nombre* **from** *asignatura*)

sin utilizar el constructor **unique.**

3.23. Considere la consulta:

select *asignatura_id*, *semestre*, *año*, *secc_id*, **avg** (*tot_créditos*)
from *matricula* **natural join** *estudiante*
where *año* = 2009
group by *asignatura_id*, *semestre*, *año*, *secc_id*
having count (*ID*) >= 2;

Explique por qué unir la *sección* en la cláusula **from** no modificaría el resultado.

3.24. Considere la consulta:

with *dept_total* (*nombre_dept*, *valor*) **as**
(**select** *nombre_dept*, **sum**(*sueldo*)
from *profesor*
group by *nombre_dept*),
dept_total_medio(*valor*) **as**
(**select avg**(*valor*)
from *dept_total*)
select *nombre_dept*
from *dept_total*, *dept_total_medio*
where *dept_total.valor* >= *dept_total_medio.valor*;

Reescriba esta consulta sin utilizar el constructor **with.**

miembro(miem_num, nombre, edad)
libro(isbn, nombre, autores, editorial)
prestado(miem_num, isbn, fecha)

Figura 3.21. Base de datos para el Ejercicio 3.21.

Herramientas

Existen diversos sistemas de bases de datos relacionales comerciales, incluyendo IBM DB2, IBM Informix, Oracle, Sybase y Microsoft SQL Server. Además, también se pueden descargar y utilizar libremente sin cargo varios sistemas de bases de datos, incluyendo PostgreSQL, MySQL (de uso libre excepto para ciertos tipos de usos comerciales) y Oracle Express edition.

La mayoría de los sistemas de bases de datos proporcionan una interfaz de línea de comandos para ejecutar comandos SQL. Además, la mayoría de los sistemas de bases de datos proporcionan una interfaz gráfica de usuario (GUI) que simplifican la tarea de navegación por la base de datos, la creación y ejecución de consultas y la administración de la misma. Entre los entornos de desarrollo comerciales para SQL que funcionan con múltiples plataformas de bases de datos están RAD Studio de Embarcadero y Aqua Data Studio.

Para PostgreSQL, la herramienta pgAdmin proporciona la funcionalidad de un GUI, mientras que para MySQL, es phpMyAdmin. NetBeans ofrece un *front-end* gráfico que funciona con distintas bases de datos pero con una funcionalidad limitada. Mientras que Eclipse proporciona una funcionalidad similar utilizando diversas extensiones como Data Tools Platform (DTP) y JBuilder.

En el sitio web puede encontrar las definiciones del esquema de datos SQL y datos de ejemplo para el esquema de la universidad. El sitio web también contiene instrucciones sobre cómo configurar y acceder a distintos sistemas de bases de datos. Los constructores de SQL que se tratan en este capítulo forman parte de la norma de SQL, pero algunas características no están disponibles en ciertas bases de datos. En el sitio web tiene una lista de estas incompatibilidades, que tendrá que tener en cuenta cuando ejecute las consultas en dichas bases de datos.

Notas bibliográficas

La versión original de SQL, denominada Sequel 2, se describe en Chamberlin et ál. [1976]. Sequel 2 procede de los lenguajes Square (Boyce et ál. [1975] y Chamberlin y Boyce [1974]). La norma American National Standard SQL-86 se describe en ANSI [1986]. La definición de SQL según la Arquitectura de Aplicación de Sistemas (System Application Architecture) de IBM se describe en IBM [1987]. Las normas oficiales de SQL-89 y de SQL-92 están disponibles en ANSI [1989] y ANSI [1992], respectivamente.

Entre las descripciones del lenguaje SQL-92 en libros de texto están Date y Darwen [1997], Melton y Simon [1993] y Cannan y Otten [1993]. Date y Darwen [1997] y Date [1993a] incluyen una crítica de SQL-92 desde la perspectiva de los lenguajes de programación.

Entre los libros de texto sobre SQL:1999 están Melton y Simon [2001] y Melton [2002]. Eisenberg y Melton [1999] ofrecen una visión general de SQL:1999. Donahoo y Speegle [2005] abordan SQL desde el punto de vista de los desarrolladores. Eisenberg et ál. [2004] ofrecen una visión general de SQL:2003.

Las normas SQL:1999, SQL:2003, SQL:2006 y SQL:2008 están publicadas como conjuntos de documentos de normas ISO/IEC, que se describen con más detalle en la Sección 24.4. Los documentos de normas tienen gran densidad de información, resultan difíciles de leer y son útiles, sobre todo para los implementadores de bases de datos. Los documentos de normas están disponibles para su compra electrónica en el sitio web http://webstore.ansi.org.

Muchos productos de bases de datos soportan más características de SQL que las especificadas en las normas, y puede que no soporten algunas características de la norma. Se puede encontrar más información sobre estas características en los manuales de usuario de SQL de los productos respectivos.

El procesamiento de las consultas SQL, incluidos los algoritmos y las consideraciones sobre rendimiento, se estudia en los Capítulos 12 y 13. En esos mismos capítulos hay referencias bibliográficas al respecto.

SQL intermedio

04

En este capítulo se continúa con el estudio de SQL. Se van a considerar formas más complejas de las consultas de SQL, la definición de vistas, las transacciones, las restricciones de integridad, más detalles sobre la definición de datos de SQL y la autorización.

4.1. Expresiones de reunión

En la Sección 3.3.3 se presentó la operación de **reunión natural.** SQL proporciona otras formas de la operación de reunión, como la posibilidad de especificar explícitamente **predicados de reunión,** y la posibilidad de incluir en el resultado tuplas que excluyen la **reunión natural.** En esta sección se tratarán estas formas de reunión.

Los ejemplos de esta sección se realizan sobre las dos relaciones *estudiante* y *matricula*, que se muestran en la Figura 4.1 y la Figura 4.2, respectivamente. Observe que el atributo *nota* tiene un valor *null* en el estudiante con *ID* 98988, en la asignatura BIO-301, sección 1, impartida en el verano de 2010. El valor *null* indica que todavía no se ha puesto la nota.

4.1.1. Condiciones de reunión

En la Sección 3.3.3 se vio cómo expresar la reunión natural y cómo utilizar la cláusula **join... using,** que es una forma de reunión natural que solo requiere que los valores sobre determinados atributos casen. SQL soporta otras formas de reunión, en las que se pueden especificar condiciones arbitrarias de reunión.

La condición **on** permite la reunión de un predicado general sobre relaciones. Este predicado se escribe como un predicado de la cláusula **where** excepto por el uso de la palabra clave **on** en lugar de **where.** Al igual que la condición **using,** la condición **on** aparece al final de la expresión de reunión.

Considere la siguiente consulta, que es una expresión de reunión que contiene la condición **on:**

select *
from *estudiante* **join** *matricula* **on** *estudiante.ID* = *matricula.ID*;

La condición **on** anterior especifica que las tuplas de *estudiante* casan con las tuplas de *matricula* si sus valores de *ID* son iguales. La expresión de reunión en este caso es casi la misma que la de reunión *estudiante* **natural join** *matricula*, ya que la reunión natural también requiere que casen las tuplas de *estudiante* con las de *matricula*. La diferencia es que el resultado contiene los atributos *ID* dos veces en la reunión, una vez por la parte *estudiante* y otra vez por *matricula*, aunque sus valores de *ID* sean los mismos.

ID	*nombre*	*nombre_dept*	*tot_créditos*
00128	Zhang	Informática	102
12345	Shankar	Informática	32
19991	Brandt	Historia	80
23121	Chávez	Finanzas	110
44553	Peltier	Física	56
45678	Levy	Física	46
54321	Williams	Informática	54
55739	Sánchez	Música	38
70557	Snow	Física	0
76543	Brown	Informática	58
76653	Aoi	Electrónica	60
98765	Bourikas	Electrónica	98
98988	Tanaka	Biología	120

Figura 4.1. La relación *estudiante*.

ID	*asignatura_id*	*secc_id*	*semestre*	*año*	*nota*
00128	CS-101	1	Otoño	2009	A
00128	CS-347	1	Otoño	2009	A−
12345	CS-101	1	Otoño	2009	C
12345	CS-190	2	Primavera	2009	A
12345	CS-315	1	Primavera	2010	A
12345	CS-347	1	Otoño	2009	A
19991	HIS-351	1	Primavera	2010	B
23121	FIN-201	1	Primavera	2010	C+
44553	PHY-101	1	Otoño	2009	B−
45678	CS-101	1	Otoño	2009	F
45678	CS-101	1	Primavera	2010	B+
45678	CS-319	1	Primavera	2010	B
54321	CS-101	1	Otoño	2009	A−
54321	CS-190	2	Primavera	2009	B+
55739	MU-199	1	Primavera	2010	A−
76543	CS-101	1	Otoño	2009	A
76543	CS-319	2	Primavera	2010	A
76653	EE-181	1	Primavera	2009	C
98765	CS-101	1	Otoño	2009	C−
98765	CS-315	1	Primavera	2010	B
98988	BIO-101	1	Verano	2009	A
98988	BIO-301	1	Verano	2010	*null*

Figura 4.2. La relación *matricula*.

De hecho, la consulta anterior es equivalente a la siguiente (en otras palabras, generan exactamente los mismos resultados):

select *
from *estudiante*, *matricula*
where *estudiante.ID* = *matricula.ID*;

Como se ha visto anteriormente, se usa el nombre de la relación para eliminar la ambigüedad del nombre del atributo *ID* y, por tanto, que pueda hablarse de *estudiante.ID* y *matricula.ID*, respectivamente. Una versión de esta consulta que muestra los valores de *ID* solo una vez es la siguiente:

select *estudiante.ID* **as** *ID*, *nombre*, *nombre_dept*, *tot_créditos*, *asignatura_id*, *secc_id*, *semestre*, *año*, *nota*
from *estudiante* **join** *matricula* **on** *estudiante.ID* = *matricula.ID*;

El resultado de la consulta anterior se muestra en la Figura 4.3.

La condición **on** puede expresar cualquier predicado de SQL y, por tanto, una expresión de reunión que incluya la condición **on** puede expresar una clase de reunión más rica que la **reunión natural.** Sin embargo, como ilustraba el ejemplo anterior, una consulta que emplee una expresión de reunión con una condición **on** se puede sustituir por una expresión equivalente sin la condición **on** siempre que el predicado de la cláusula **on** sea desplazado a la cláusula **where.** Por tanto, puede parecer que la condición **on** es una característica redundante de SQL.

Sin embargo, existen dos buenas razones para introducir la condición **on.** La primera se verá en breve para ciertos tipos de reuniones denominadas reuniones externas, la condición **on** se comporta de manera diferente a la condición **where.** La segunda es que una consulta de SQL generalmente es más legible para las personas si las condiciones de reunión se especifican en la cláusula **on** y el resto de las condiciones aparecen en la cláusula **where.**

4.1.2. Reunión externa

Suponga que se desea mostrar una lista de todos los estudiantes, con sus *ID* respectivos y el *nombre*, *nombre_dept* y *tot_créditos*, junto con las asignaturas en las que se han matriculado. La siguiente consulta de SQL podría parecer que obtiene la información deseada:

select *
from *estudiante* **natural join** *matricula*;

Desafortunadamente, la consulta anterior no funciona como se pretende. Suponga que existe algún estudiante que no se matricula en ninguna asignatura. Entonces la tupla de la relación estudiante para ese estudiante no satisfaría la condición de una reunión natural con ninguna tupla de la relación matricula, y ese estudiante no aparecería en los resultados. Por tanto, no se vería ninguna información de los estudiantes que no se han matriculado en ninguna asignatura. Por ejemplo, en las relaciones estudiante y matricula de la Figura 4.1 y la Figura 4.2 se puede ver que el estudiante Show, con ID 70557, no se ha matriculado en ninguna asignatura. Por tanto, Show no aparece en el resultado de la reunión natural.

De forma más general, al realizar la reunión algunas de las tuplas de una de las relaciones, o de ambas, se «pierden». La operación **reunión externa** (*outer join*) funciona de forma similar a la operación de reunión, creando tuplas en el resultado que contienen valores nulos.

Por ejemplo, para asegurarse de que el estudiante de nombre Snow del ejemplo anterior aparece en el resultado, se podría añadir una tupla al resultado de la reunión con todos los atributos de la relación *estudiante* que se corresponden con los del estudiante Snow, y el resto de atributos provienen de la relación *matricula*, es decir *asignatura_id*, *secc_id*, *semestre* y *año*, puestos a *null*. Por

ID	*nombre*	*nombre_dept*	*tot_créditos*	*asignatura_id*	*secc_id*	*semestre*	*año*	*nota*
00128	Zhang	Informática	102	CS-101	1	Otoño	2009	A
00128	Zhang	Informática	102	CS-347	1	Otoño	2009	A–
12345	Shankar	Informática	32	CS-101	1	Otoño	2009	C
12345	Shankar	Informática	32	CS-190	2	Primavera	2009	A
12345	Shankar	Informática	32	CS-315	1	Primavera	2010	A
12345	Shankar	Informática	32	CS-347	1	Otoño	2009	A
19991	Brandt	Historia	80	HIS-351	1	Primavera	2010	B
23121	Chávez	Finanzas	110	FIN-201	1	Primavera	2010	C+
44553	Peltier	Física	56	PHY-101	1	Otoño	2009	B–
45678	Levy	Física	46	CS-101	1	Otoño	2009	F
45678	Levy	Física	46	CS-101	1	Primavera	2010	B+
45678	Levy	Física	46	CS-319	1	Primavera	2010	B
54321	Williams	Informática	54	CS-101	1	Otoño	2009	A–
54321	Williams	Informática	54	CS-190	2	Primavera	2009	B+
55739	Sánchez	Música	38	MU-199	1	Primavera	2010	A–
76543	Brown	Informática	58	CS-101	1	Otoño	2009	A
76543	Brown	Informática	58	CS-319	2	Primavera	2010	A
76653	Aoi	Electrónica	60	EE-181	1	Primavera	2009	C
98765	Bourikas	Electrónica	98	CS-101	1	Otoño	2009	C–
98765	Bourikas	Electrónica	98	CS-315	1	Primavera	2010	B
98988	Tanaka	Biología	120	BIO-101	1	Verano	2009	A
98988	Tanaka	Biología	120	BIO-301	1	Verano	2010	*null*

Figura 4.3. Resultado de *estudiante* **join** *matricula* **on** *estudiante.ID* = *matricula.ID* en el que se ha omitido la segunda ocurrencia de *ID*.

tanto, la tupla del estudiante Snow se conserva en el resultado de la reunión externa.

De hecho existen tres formas de reunión externa:

- La **reunión externa izquierda** (*left outer join*) conserva las tuplas solo en la relación de antes (a la izquierda) de la operación de **reunión externa izquierda.**
- La **reunión externa derecha** (*right outer join*) conserva las tuplas solo en la relación de después (a la derecha) de la operación de **reunión externa derecha.**
- La **reunión externa completa** (*full outer join*) conserva las tuplas de ambas relaciones.

En contraste, las operaciones de reunión que se trataron anteriormente y que no conservaban las tuplas que no casaban se denominan operaciones de **reunión interna,** para distinguirlas de las operaciones de reunión externa.

A continuación se va a ver cómo funciona exactamente cada una de las formas de reunión externa. Se puede calcular la operación de reunión externa como sigue: primero se calcula el resultado de la reunión interna como anteriormente; después, para cada tupla *t* en la parte izquierda de la relación que no casa con ninguna tupla de la parte derecha en una reunión interna, se añade una tupla *r* al resultado, que se construye de la siguiente forma:

- Los atributos de la tupla *r* que se derivan de la relación de la parte izquierda se rellenan con los valores de la tupla *t*.
- El resto de atributos de *r* se rellenan con valores *null*.

En la Figura 4.4 se muestra el resultado de:

select *
from *estudiante* **natural left outer join** *matricula*;

El resultado incluye al estudiante Snow (*ID* 70557), al contrario que una reunión interna, pero la tupla de Snow incluye valores *null* para todos los atributos que solo aparecen en el esquema de la relación *matricula*.

Como otro ejemplo del uso de la operación de reunión externa, se puede escribir la consulta: «Encontrar a todos los estudiantes que no se hayan matriculado en ninguna asignatura» como:

select *ID*
from *estudiante* **natural left outer join** *matricula*
where *asignatura_id* **is** *null*;

La **reunión externa derecha** es simétrica a la **reunión externa izquierda.** Las tuplas de la derecha de la relación que no casan con ninguna tupla de la izquierda se rellenan con *null* y se añaden al resultado de la reunión externa. Por tanto, si se reescribiese la consulta anterior usando una reunión externa derecha e intercambiando el orden en que se indican las relaciones de la siguiente forma:

select *
from *matricula* **natural right outer join** *estudiante*;

se obtendría el mismo resultado, excepto por el orden en que aparecen los atributos en el resultado (véase la Figura 4.5).

La **reunión externa completa** es una combinación de los tipos de reunión externa derecha e izquierda. Después de que la operación calcule el resultado de la reunión interna, extiende con valores *null* aquellas tuplas de la parte izquierda de la reunión que no casan con ninguna de la parte derecha y las añade al resultado. De forma similar, extiende con valores *null* aquellas tuplas de la parte derecha que no casan con tuplas de la parte izquierda y las añade al resultado.

ID	*nombre*	*nombre_dept*	*tot_créditos*	*asignatura_id*	*secc_id*	*semestre*	*año*	*nota*
00128	Zhang	Informática	102	CS-101	1	Otoño	2009	A
00128	Zhang	Informática	102	CS-347	1	Otoño	2009	A–
12345	Shankar	Informática	32	CS-101	1	Otoño	2009	C
12345	Shankar	Informática	32	CS-190	2	Primavera	2009	A
12345	Shankar	Informática	32	CS-315	1	Primavera	2010	A
12345	Shankar	Informática	32	CS-347	1	Otoño	2009	A
19991	Brandt	Historia	80	HIS-351	1	Primavera	2010	B
23121	Chávez	Finanzas	110	FIN-201	1	Primavera	2010	C+
44553	Peltier	Física	56	PHY-101	1	Otoño	2009	B–
45678	Levy	Física	46	CS-101	1	Otoño	2009	F
45678	Levy	Física	46	CS-101	1	Primavera	2010	B+
45678	Levy	Física	46	CS-319	1	Primavera	2010	B
54321	Williams	Informática	54	CS-101	1	Otoño	2009	A–
54321	Williams	Informática	54	CS-190	2	Primavera	2009	B+
55739	Sánchez	Música	38	MU-199	1	Primavera	2010	A–
70557	Snow	Física	0	*null*	*null*	*null*	*null*	*null*
76543	Brown	Informática	58	CS-101	1	Otoño	2009	A
76543	Brown	Informática	58	CS-319	2	Primavera	2010	A
76653	Aoi	Electrónica	60	EE-181	1	Primavera	2009	C
98765	Bourikas	Electrónica	98	CS-101	1	Otoño	2009	C–
98765	Bourikas	Electrónica	98	CS-315	1	Primavera	2010	B
98988	Tanaka	Biología	120	BIO-101	1	Verano	2009	A
98988	Tanaka	Biología	120	BIO-301	1	Verano	2010	*null*

Figura 4.4. Resultado de *estudiante* **natural left outer join** *matricula*.

ID	*asignatura_id*	*secc_id*	*semestre*	*año*	*nota*	*nombre*	*nombre_dept*	*tot_cred*
00128	CS-101	1	Otoño	2009	A	Zhang	Informática	102
00128	CS-347	1	Otoño	2009	A–	Zhang	Informática	102
12345	CS-101	1	Otoño	2009	C	Shankar	Informática	32
12345	CS-190	2	Primavera	2009	A	Shankar	Informática	32
12345	CS-315	1	Primavera	2010	A	Shankar	Informática	32
12345	CS-347	1	Otoño	2009	A	Shankar	Informática	32
19991	HIS-351	1	Primavera	2010	B	Brandt	Historia	80
23121	FIN-201	1	Primavera	2010	C+	Chávez	Finanzas	110
44553	PHY-101	1	Otoño	2009	B–	Peltier	Física	56
45678	CS-101	1	Otoño	2009	F	Levy	Física	46
45678	CS-101	1	Primavera	2010	B+	Levy	Física	46
45678	CS-319	1	Primavera	2010	B	Levy	Física	46
54321	CS-101	1	Otoño	2009	A–	Williams	Informática	54
54321	CS-190	2	Primavera	2009	B+	Williams	Informática	54
55739	MU-199	1	Primavera	2010	A–	Sánchez	Música	38
70557	*null*	*null*	*null*	*null*	*null*	Snow	Física	0
76543	CS-101	1	Otoño	2009	A	Brown	Informática	58
76543	CS-319	2	Primavera	2010	A	Brown	Informática	58
76653	EE-181	1	Primavera	2009	C	Aoi	Electrónica	60
98765	CS-101	1	Otoño	2009	C–	Bourikas	Electrónica	98
98765	CS-315	1	Primavera	2010	B	Bourikas	Electrónica	98
98988	BIO-101	1	Verano	2009	A	Tanaka	Biología	120
98988	BIO-301	1	Verano	2010	*null*	Tanaka	Biología	120

Figura 4.5. Resultado de *matricula* **natural right outer join** *estudiante*.

Como un ejemplo del uso de la reunión externa completa, considere la siguiente consulta: «Mostrar una lista de todos los estudiantes del departamento de Informática, junto con las secciones de asignatura, si existen, que se han enseñado en la primavera de 2009; se deben mostrar todas las secciones de la primavera de 2009, incluso aunque ningún estudiante del departamento de Informática se haya matriculado en la sección». Esta consulta se puede escribir como:

```
select *
from (select *
      from estudiante
      where nombre_dept = 'Informática')
     natural full outer join
     (select *
      from matricula
      where semestre = 'Primavera' and año = 2009);
```

Se puede usar la cláusula **on** con las reuniones externas. La siguiente consulta es idéntica a la primera que se vio «*estudiante* **natural left outer join** *matricula*», excepto que el atributo *ID* aparece dos veces en el resultado.

```
select *
from estudiante left outer join matricula
      on estudiante.ID = matricula.ID;
```

Como se ha indicado anteriormente, **on** y **where** se comportan de forma diferente en la reunión externa. La razón es que la reunión externa solo rellena con valores *null* las tuplas que no contribuyen al resultado de la correspondiente reunión interna. La condición **on** forma parte de la especificación de la reunión externa, pero la cláusula **where** no. En nuestro ejemplo, el caso de la tupla *estudiante* para el estudiante «Snow», con ID 70557, muestra esta distinción. Suponga que se modifica la consulta anterior trasladando la cláusula **where** y en su lugar se usa la condición **on** a *true*.

```
select *
from estudiante left outer join matricula on true
where estudiante.ID = matricula.ID;
```

La consulta anterior, usando la reunión externa izquierda con la condición on, incluye una tupla (70557, Snow, Física, , null, null, null, null, null, null), porque no existe ninguna tupla en matricula con ID = 70557. En la última consulta, sin embargo, todas las tuplas satisfacen la condición de reunión, por lo que no se generan tuplas rellenas con *null* con la reunión externa. La reunión externa realmente genera el producto cartesiano de las dos relaciones. Como no existe tupla en matricula con ID = 70557, cada vez que aparece una tupla en la reunión externa con nombre = «Snow», los valores de estudiante.ID y matricula.ID deben ser diferentes y, por tanto, los debería eliminar el predicado de la cláusula **where.** Por tanto, el estudiante Snow nunca aparece en el resultado de la última consulta.

Tipos de reunión
inner join
left outer join
right outer join
full outer join

Condiciones de reunión
natural
on <predicado>
using (A_1, A_2, …, A_n)

Figura 4.6. Tipos y condiciones de reunión.

4.1.3. Tipos de reunión y condiciones

Para distinguir las reuniones normales de las externas, a las normales se les denomina **reunión interna** en SQL. Por tanto, una cláusula de reunión puede especificar una **inner join** (reunión interna), en lugar de una **outer join** (reunión externa) para especificar que se usa una reunión normal. La palabra clave **inner,** sin embargo, es opcional. El tipo de reunión, por defecto, cuando se usa la cláusula **join** sin el prefijo **outer,** es la reunión interna **(inner join).** Por tanto:

select *
from *estudiante* **join** *matricula* **using** (*ID*);

es equivalente a:

select *
from *estudiante* **inner join** *matricula* **using** (*ID*);

De la misma forma, **natural join** (reunión natural) es equivalente a **natural inner join** (reunión natural interna).

La Figura 4.6 muestra una lista completa de los distintos tipos de reunión que se han tratado. Se observa que cualquier forma de reunión (interna, externa izquierda, externa derecha o completa) se puede combinar con cualquier condición de reunión (*natural*, *using*, *on*).

4.2. Vistas

En nuestros ejemplos, hasta este momento se ha trabajado en el nivel del modelo lógico. Es decir, se ha supuesto que las relaciones de la colección que se ha utilizado son las relaciones reales almacenadas en la base de datos.

No es deseable que todos los usuarios puedan ver el modelo lógico completo. Por razones de seguridad se puede requerir que ciertos datos de la base de datos queden ocultos a los usuarios. Considere un empleado que necesita conocer el ID de un profesor, su nombre y el nombre de su departamento, pero no tiene autorización para ver el sueldo del profesor. Esta persona debería ver una relación descrita en SQL como:

select *ID*, *nombre*, *nombre_dept*
from *instructor*;

Junto a las razones de seguridad, se puede desear crear una colección personalizada de relaciones que casen mejor con la intuición del usuario que el propio modelo lógico. Se puede desear disponer de una lista de todas las secciones de asignaturas que se ofertaron en el departamento de Física en el semestre del otoño de 2009, junto con el edificio y el número de aula de cada sección. La relación que se debería crear para obtener esta lista es:

select *asignatura.asignatura_id*, *secc_id*, *edificio*, *número_aula*
from *asignatura*, *sección*
where *asignatura.asignatura_id* = *sección.asignatura_id*
and *asignatura.nombre_dept* = ´Física´
and *sección.semestre* = ´Otoño´
and *sección.año* = ´2009´;

Es posible calcular y almacenar el resultado de la consulta anterior y después poner a disposición de los usuarios la relación almacenada. Sin embargo, si se hiciese así, y los datos subyacentes de las relaciones *profesor*, *asignatura* o *sección* cambiasen, los resultados de la consulta almacenados ya no coincidirían con el resultado de volver a ejecutar la consulta sobre las relaciones. En general, es una mala idea calcular y almacenar el resultado de una consulta como en el ejemplo anterior (aunque existen algunos casos excepcionales, como se verá más adelante).

En lugar de ello, SQL permite definir una «relación virtual» y que contenga conceptualmente el resultado de la consulta. La relación virtual no se precalcula y almacena, sino que se calcula ejecutando la consulta cuando la relación virtual se utiliza.

Una relación como esta que no forma parte del modelo lógico pero se hace visible a los usuarios como una relación virtual se denomina **vista.** Es posible manejar un gran número de vistas sobre cualquier conjunto de relaciones reales.

4.2.1. Definición de vistas

En SQL se define una vista usando el comando **create view.** Para definir una vista se debe dar un nombre a la vista e indicar la consulta que calcula la vista. El formato del comando **create view** es:

create view *v* **as** <expresión de consulta>;

donde <expresión de consulta> es cualquier expresión legal de consulta. El nombre de la vista se representa mediante *v*.

Considere de nuevo al empleado que necesita acceder a todos los datos de la relación *profesor*, excepto el sueldo. El empleado no debería tener autorización para acceder a la relación *profesor* (más adelante se verá, en la Sección 4.6, cómo especificar autorizaciones). En su lugar, se puede poner a su disposición una relación de vista *facultad* para el empleado, con la vista definida de la siguiente forma:

create view *facultad* **as**
select *ID*, *nombre*, *nombre_dept*
from *profesor*;

Como se ha explicado anteriormente, la relación de vista contiene conceptualmente todas las tuplas del resultado de la consulta, pero no es precalculada ni almacenada. En su lugar, el sistema de bases de datos almacena la expresión de la consulta asociada a la relación de la vista. Siempre que se accede a la relación de la vista se crea cuando se necesita, bajo demanda.

Para crear una vista con la lista de todas las secciones de asignaturas que ofertó el departamento de Física en el semestre del otoño de 2009 con su edificio y el nombre del aula de cada sección, se escribirá:

create view *física_otoño_2009* **as**
select *asignatura.asignatura_id*, *secc_id*, *edificio*, *número_aula*
from *asignatura*, *sección*
where *asignatura.asignatura_id* = *sección.asignatura_id*
and *asignatura.nombre_dept* = ´Física´
and *sección.semestre* = ´Otoño´
and *sección.año* = ´2009´;

4.2.2. Uso de vistas en las consultas de SQL

Una vez definida la vista se puede utilizar su nombre para hacer referencia a la relación virtual que genera. Utilizando la vista *física_otoño_2009* se pueden encontrar todas las asignaturas de Física que se ofertaron en el semestre del otoño de 2009 en el edificio Watson, escribiendo:

select *asignatura_id*
from *física_otoño_2009*
where *edificio* = ´Watson´;

Los nombres de las vistas pueden aparecer en cualquier lugar en el que puedan hacerlo los nombres de las relaciones.

Los nombres de los atributos de las vistas se pueden especificar explícitamente de la siguiente forma:

create view *departamentos_total_sueldos*(*nombre_dept*, *total_sueldo*) **as**
select *nombre_dept*, **sum** (*sueldo*)
from *profesor*
group by *nombre_dept*;

La vista anterior da para cada departamento la suma de los sueldos de todos los profesores del departamento. Como la expresión **sum**(*sueldo*) no tiene nombre, el nombre del atributo se especifica explícitamente en la definición de la vista.

De manera intuitiva, el conjunto de tuplas de la relación de vistas es el resultado de la evaluación de la expresión de consulta que define la vista. Por tanto, si una relación de vistas se calcula y se almacena, pueden quedar desfasadas si se modifican las relaciones usadas para definirlas. Para evitarlo, las vistas suelen implementarse de la siguiente forma. Cuando se define una vista, el sistema de la base de datos guarda la definición de la vista, en vez del resultado de la evaluación de la expresión que define. Siempre que aparece una relación de vista en una consulta, se sustituye por la expresión de consulta almacenada. Por tanto, la relación de vista se vuelve a calcular siempre que se evalúa la consulta.

Se puede utilizar una vista en una expresión que define otra vista. Por ejemplo, se puede definir una vista *física_otoño_2009_watson*, que lista los id de asignaturas y los números de aula de todas las asignaturas de Física que se ofertaron en el semestre de 2009 en el edificio Watson de la siguiente forma:

```
create view física_otoño_2009_watson as
        select asignatura_id, número_aula
        from física_otoño_2009
        where edificio = ´Watson´;
```

donde *física_otoño_2009_watson* es una relación de vista. Es equivalente a:

```
create view física_otoño_2009_watson as
    (select asignatura_id, número_aula
     from (select asignatura.asignatura_id, edificio,
                  número_aula
           from asignatura, sección
           where asignatura.asignatura_id = sección.
                 asignatura_id
                 and asignatura.nombre_dept = ´Física´
                 and sección.semestre = ´Otoño´
                 and sección.año = ´2009´)
     where edificio = ´Watson´;
```

4.2.3. Vistas materializadas

Algunos sistemas de bases de datos permiten que se guarden las relaciones de vista, pero se aseguran de que si las relaciones reales utilizadas en la definición de la vista cambian, la vista se mantenga actualizada. Estas vistas se denominan **vistas materializadas.**

Por ejemplo, considere la vista *departamentos_total_sueldos*. Si esta vista es materializada, su resultado se debería guardar en la base de datos. Sin embargo, si se añade o se borra una tupla de la relación *profesor*, el resultado de la consulta que define la vista debería cambiar, y se debería actualizar el contenido de la vista materializada. De forma similar, si se actualiza el sueldo de un profesor, la tupla en *departamentos_total_sueldos* que corresponde al departamento de dicho profesor también se debería actualizar.

El proceso de mantener actualizada la vista se denomina **mantenimiento de vistas materializadas** (o solamente **mantenimiento de vistas**) y se trata en la Sección 13.5. El mantenimiento de vistas se puede realizar inmediatamente después de que cualquiera de las relaciones sobre las que se define la vista se haya actualizado. Algunas bases de datos, sin embargo, realizan el mantenimiento de vistas de forma perezosa, cuando se accede a la misma. Otros sistemas de bases de datos actualizan las vistas materializadas solo periódicamente; en este caso, el contenido de la vista materializada puede estar desfasado, es decir, no estar actualizado. Y también otras bases de datos permiten que el administrador controle cuál de los métodos anteriores se utilizará para cada vista materializada.

Las aplicaciones en las que se utiliza frecuentemente una vista pueden aprovechar las vistas materializadas. Las aplicaciones que exigen una respuesta rápida a ciertas consultas que calculan agregados de grandes relaciones también pueden aprovechar enormemente la creación de vistas materializadas correspondientes a las consultas. En este caso los resultados agregados serán probablemente mucho más pequeños que las grandes relaciones sobre las que se definen; por ello la vista materializada se puede utilizar para responder a una consulta muy rápidamente, evitando la lectura de las grandes relaciones subyacentes. Por supuesto, las ventajas para las consultas derivadas de la materialización de las vistas deben sopesarse frente a los costes de almacenamiento y la sobrecarga añadida de las actualizaciones.

SQL no define ninguna forma estándar de especificar que una vista es materializada, pero los sistemas de bases de datos proporcionan sus propias extensiones de SQL para esta tarea. Algunos sistemas de bases de datos siempre mantienen las vistas materializadas actualizadas cuando las relaciones subyacentes cambian, mientras que otros sistemas permiten que se queden desactualizadas y las recalculan periódicamente.

4.2.4. Actualización de vistas

Aunque las vistas resultan una herramienta útil para las consultas, presentan serios problemas si se expresan con ellas actualizaciones, inserciones o borrados. La dificultad radica en que las modificaciones de la base de datos expresadas en términos de vistas deben traducirse en modificaciones de las relaciones reales del modelo lógico de la base de datos.

Suponga la vista *facultad*, que se vio anteriormente, que se pone a disposición del empleado. Como se permite que el nombre de la vista aparezca en cualquier lugar en el que pueda aparecer el nombre de una relación, el empleado puede escribir:

```
insert into facultad
values (´30765´, ´Green´, ´Música´);
```

Esta inserción debe representarse mediante una inserción en la relación *profesor*, puesto que *profesor* es la relación real a partir de la cual el sistema de bases de datos construye la vista *facultad*. Sin embargo, para insertar una tupla en *profesor* hay que tener algún valor para *sueldo*. Hay dos enfoques razonables para tratar esta inserción:

- Rechazar la inserción y devolver al usuario un mensaje de error.
- Insertar la tupla (´30765´, ´Green´, ´Música´, *null*) en la relación *profesor*.

Otro problema con la modificación de la base de datos mediante vistas surge con vistas como:

```
create view profesor_info as
select ID, nombre, edificio
from profesor, departamento
where profesor.nombre_dept = departamento.nombre_dept;
```

Esta vista muestra el *ID*, el *nombre* y el *edificio* de todos los profesores de la universidad. Considere la siguiente inserción mediante esta vista:

```
insert into profesor_info
        values (´69987´, ´White´, ´Taylor´);
```

Suponga que no existe ningún profesor con ID 69987, ni ningún departamento en el edificio Taylor. Entonces el único método posible de insertar tuplas en las relaciones *profesor* y *departamento* es insertar (´69987´, ´White´, *null*, *null*) en *profesor* y (*null*, ´Taylor´, *null*) en *departamento*. Entonces se obtienen las relaciones que se muestran en la Figura 4.7. Sin embargo, esta actualización no ge-

nera el efecto deseado, ya que la relación de vista *profesor_info* no contiene todavía la tupla (´69987´, ´White´, ´Taylor´). Por tanto, no existe ninguna forma de actualizar las relaciones *profesor* y *departamento* usando valores *null* para conseguir actualizar *profesor_info*.

Debido a problemas como estos no se suelen permitir las actualizaciones de las relaciones de vistas, salvo en casos concretos. Los diferentes sistemas de bases de datos especifican condiciones diferentes para permitir las actualizaciones de las relaciones de vistas y es preciso consultar el manual de cada sistema para conocer los detalles. El problema general de la modificación de las bases de datos mediante las vistas ha sido objeto de amplia investigación, y las notas bibliográficas ofrecen indicaciones de parte de esa investigación.

En general se dice que una vista de SQL es **actualizable** (es decir, se pueden aplicar inserciones, actualizaciones y borrados) si se cumplen todas las condiciones siguientes:

- La cláusula **from** solo tiene una relación de bases de datos.
- La cláusula **select** solo contiene nombres de atributos de la relación y no tiene ninguna expresión, valor agregado ni especificación **distinct.**
- Cualquier atributo que no aparezca en la cláusula **select** puede definirse como *null*, es decir, no contiene ninguna restricción **not null** y no forma parte de una clave primaria.
- La consulta no tiene cláusulas **group by** ni **having.**

Con estas restricciones, las operaciones **update, insert** y **delete** podrían permitirse en la siguiente vista:

create view *historia_profesores* **as**
select *
from *profesor*
where *nombre_dept* = ´Historia´;

ID	*nombre*	*nombre_dept*	*sueldo*
10101	Srinivasan	Informática	65000
12121	Wu	Finanzas	90000
15151	Mozart	Música	40000
22222	Einstein	Física	95000
32343	El Said	Historia	60000
33456	Gold	Física	87000
45565	Katz	Informática	75000
58583	Califieri	Historia	62000
76543	Singh	Finanzas	80000
76766	Crick	Biología	72000
83821	Brandt	Informática	92000
98345	Kim	Electrónica	80000
69987	White	*null*	*null*

Profesor

nombre_dept	*edificio*	*presupuesto*
Biología	Watson	90000
Informática	Taylor	100000
Electrónica	Taylor	85000
Finanzas	Painter	120000
Historia	Painter	50000
Música	Packard	80000
Física	Watson	70000
Null	Painter	*null*

Departamento

Figura 4.7. Relaciones *profesor* y *departamento* tras la inserción de tuplas.

Incluso con estas condiciones para la actualización, sigue existiendo el siguiente problema. Suponga que un usuario intenta insertar la tupla (´25566´, ´Brown´, ´Biología´, 100000) en la vista *historia_profesores*. Esta tupla puede insertarse en la relación *profesor*, pero no aparecerá en la vista *historia_profesores*, ya que no cumple la selección impuesta por la vista.

De manera predeterminada, SQL permitiría que la actualización se llevara a cabo. Sin embargo, pueden definirse vistas con una cláusula **with check option** al final de la definición de la vista; por tanto, si una tupla insertada en la vista no cumple la condición de la cláusula **where** de la vista, el sistema de bases de datos rechaza la inserción. De manera parecida se rechazan las actualizaciones si los valores nuevos no cumplen las condiciones de la cláusula **where.**

SQL:1999 tiene un conjunto de reglas más complejo en cuanto a la posibilidad de ejecutar inserciones, actualizaciones y borrados sobre las vistas, que permite las actualizaciones en una clase más amplia de vistas; sin embargo, estas reglas son demasiado complejas para tratarlas aquí.

4.3. Transacciones

Una **transacción** consiste en una secuencia de instrucciones de consulta o de actualización. La norma SQL especifica que una transacción comienza implícitamente cuando se ejecuta una sentencia de SQL. Una de las siguientes sentencias SQL debe finalizar la transacción:

- **Commit work** (compromiso) compromete la transacción actual; es decir, hace que las actualizaciones realizadas por la transacción pasen a ser permanentes en la base de datos. Una vez comprometida la transacción, se inicia de manera automática una nueva transacción.
- **Rollback work** (retroceso) provoca el retroceso de la transacción actual; es decir, deshace todas las actualizaciones realizadas por las sentencias de SQL de la transacción. Por tanto, el estado de la base de datos se restaura al que tenía antes de la ejecución de la primera instrucción de la transacción.

La palabra clave **work** es opcional en ambas instrucciones.

El retroceso de transacciones resulta útil si se detecta alguna condición de error durante la ejecución de la transacción. El compromiso es parecido, en cierto sentido, a guardar los cambios de un documento que se esté editando, mientras que el retroceso es como abandonar la sesión de edición sin guardar los cambios. Una vez que la transacción ha ejecutado **commit work,** sus efectos ya no se pueden deshacer con **rollback work.** El sistema de bases de datos garantiza en caso de fallo (como puede ser un error en una de las instrucciones SQL, una caída de tensión o una caída del sistema) el retroceso de los efectos de la transacción si todavía no se había ejecutado **commit work.** En caso de que se produzca una caída de tensión o una caída del sistema, el retroceso se origina cuando el sistema se reinicia.

Por ejemplo, suponga una aplicación bancaria en la que se necesita transferir dinero de una cuenta a otra en el mismo banco. Para ello se necesita actualizar los saldos de dos cuentas, retirando fondos de una y añadiéndolos a la otra. Si el sistema falla tras retirar la cantidad de la primera cuenta, pero antes de añadirla a la segunda, los saldos del banco podrían quedar inconsistentes. Un problema similar podría ocurrir si se añade la cantidad en la segunda cuenta antes de retirarla de la primera y el sistema falla justo tras añadir la cantidad en la primera.

Se puede poner otro ejemplo con el caso de la aplicación de la universidad. Suponga que el atributo *tot_créditos* de las tuplas de la relación *estudiante* se mantienen actualizadas modificándo-

la siempre que el estudiante termina una asignatura. Para ello, cuando la relación *matricula* se actualiza para registrar que un estudiante ha terminado una asignatura (asignándole la correspondiente nota), también hay que actualizar la correspondiente tupla *estudiante*. Si la aplicación que realiza estas dos actualizaciones falla tras realizar una de las actualizaciones, pero antes de realizar la segunda, los datos de la base de datos quedarían inconsistentes.

Ya sea comprometiendo las acciones de una transacción tras haberse completado, o retrocediendo todas sus acciones en el caso de que la transacción no pueda completar todas sus acciones correctamente, la base de datos proporciona una abstracción de que la transacción es **atómica,** es decir, indivisible. O bien se ven reflejados todos los efectos de la transacción o no se ve ninguno (tras el retroceso).

Aplicando la noción de transacción a las aplicaciones anteriores, las sentencias de actualización se deberían ejecutar como una única transacción. Un error mientras se ejecuta una sentencia de una transacción hace que se deshagan todos los efectos de las sentencias anteriores de la transacción, de forma que en la base de datos no se mantenga ninguna actualización parcial.

Si un programa termina sin ejecutar ninguno de estos comandos, las actualizaciones se comprometen o se retroceden. La norma no especifica cuál de las dos opciones tiene lugar, con lo que la elección depende de la implementación.

En muchas implementaciones de SQL, cada instrucción de SQL se considera de manera predeterminada como una transacción en sí misma, y se compromete en cuanto se ejecuta. El compromiso automático de cada instrucción de SQL se puede desactivar si hay que ejecutar una transacción que consta de varias instrucciones SQL. La forma de desactivación del compromiso automático depende de cada implementación de SQL concreta, aunque existe una forma estándar mediante las interfaces de programación de aplicaciones como JDBC o ODBC, que se verán más adelante en las Secciones 5.1.1 y 5.1.2, respectivamente.

Una alternativa mejor, que forma parte de la norma SQL:1999 (aunque actualmente solo está soportada por algunas implementaciones de SQL) es permitir que se encierren varias instrucciones SQL entre las palabras clave **begin atomic… end.** Todas las instrucciones entre esas palabras clave forman así una única transacción.

En el Capítulo 14 se verán las propiedades de las transacciones. Los problemas de implementación en una única base de datos se tratan en los Capítulos 15 y 16, mientras que en el Capítulo 19 se tratan los problemas de implementación de transacciones entre varias bases de datos, para tratar cuestiones como la transferencia de dinero entre cuentas de distintos bancos, que tienen bases de datos diferentes.

4.4. Restricciones de integridad

Las restricciones de integridad garantizan que las modificaciones realizadas en la base de datos por los usuarios autorizados no den lugar a una pérdida de la consistencia de los datos. Por tanto, las restricciones de integridad protegen contra daños accidentales a las bases de datos.

Algunos ejemplos de restricciones de integridad son:

- El nombre de un profesor no puede ser *null*.
- No puede haber dos profesores con el mismo *ID* de profesor.
- Todos los nombres de departamento en la relación *asignatura* deben tener el nombre de departamento correspondiente en la relación *departamento*.
- El presupuesto de un departamento debe ser superior a 0.00 €.

En general, las restricciones de integridad pueden ser predicados arbitrarios que hagan referencia a la base de datos. Sin embargo, la comprobación de estos predicados arbitrarios puede resultar costosa. Por tanto, la mayor parte de los sistemas de bases de datos permiten especificar restricciones de integridad que puedan probarse con una sobrecarga mínima.

Ya se han visto algunas formas de restricciones de integridad en la Sección 3.2.2. Se verán más formas en esta sección. En el Capítulo 8 se estudia otra forma de restricción de integridad, denominada **dependencia funcional,** que se utiliza sobre todo en el proceso de diseño de esquemas.

Las restricciones de integridad suelen identificarse como parte del proceso de diseño del esquema de la base de datos, y se declaran como parte del comando **create table** utilizado para crear las relaciones. Sin embargo, también se pueden añadir a relaciones existentes utilizando el comando **alter table** *nombre-tabla* **add** *restricción*, donde *restricción* puede ser cualquier restricción sobre la relación. Cuando se ejecuta uno de estos comandos, el sistema primero se asegura que la relación satisface la restricción indicada. Si la cumple, se añade la restricción a la relación; sino, el comando se rechaza.

4.4.1. Restricciones sobre una sola relación

En la Sección 3.2 se describió la manera de definir tablas mediante el comando **create table.** Este comando también puede incluir instrucciones para restricciones de integridad. Además de la restricción «de clave primaria», hay varias más que pueden incluirse en el comando **create table.** Entre las restricciones de integridad permitidas se encuentran:

- **not null**
- **unique**
- **check**(<predicado>)

Cada uno de estos tipos de restricciones se tratan en las secciones siguientes.

4.4.2. Restricción *not null*

Como se trató en el Capítulo 3, el valor nulo (*null*) es miembro de todos los dominios y, en consecuencia, de manera predeterminada es un valor legal para todos los atributos de SQL. Para ciertos atributos, sin embargo, los valores nulos pueden resultar poco adecuados. Considere una tupla de la relación *estudiante* en la que *nombre* valga *null*. Esta tupla proporciona información de un estudiante desconocido; por tanto, no contiene información útil. De manera parecida, no es deseable que el presupuesto del departamento valga *null*. En casos así se desearía prohibir los valores nulos, lo que se puede hacer restringiendo el dominio de los atributos *nombre* y *presupuesto* para excluir los valores nulos, declarándolos de la siguiente manera:

nombre **varchar**(20) **not null**
presupuesto **numeric**(12,2) **not null**

La especificación **not null** prohíbe la inserción de valores nulos para ese atributo. Cualquier modificación de la base de datos que haga que se inserte un valor nulo en un atributo declarado como **not null** genera un diagnóstico de error.

Existen muchas situaciones en las que se desea evitar los valores nulos. En concreto, SQL prohíbe los valores nulos en la clave primaria de los esquemas de las relaciones. Por tanto, en el ejemplo de la universidad, en la relación *departamento*, si el atributo *nombre_dept* se declara clave primaria de *departamento*, no puede tener valores nulos. En consecuencia, no hace falta declararlo **not null** de manera explícita.

4.4.3. Restricción *unique*

SQL también soporta la restricción de integridad:

unique ($A_{j1}, A_{j2}, \ldots, A_{jm}$)

La especificación **unique** indica que los atributos $A_{j1}, A_{j2}, \ldots, A_{jm}$ forman una clave candidata; es decir, ningún par de tuplas de la relación puede ser igual en todos los atributos indicados. Sin embargo, se permite que los atributos de la clave candidata tengan valores *null*, a menos que se hayan declarado de manera explícita como **not null.** Recuerde que los valores nulos no son iguales a ningún otro valor. (El tratamiento de los valores nulos en este caso es el mismo que el del constructor **unique** definido en la Sección 3.8.4).

4.4.4. La cláusula *check*

Aplicada a una declaración de relación, la cláusula **check**(*P*) especifica un predicado *P* que deben satisfacer todas las tuplas de la relación.

Un uso frecuente de la cláusula **check** es garantizar que los valores de los atributos cumplan las condiciones especificadas, lo que permite en realidad construir un potente sistema de tipos. Por ejemplo, la cláusula **check**(*presupuesto* > 0) en el comando **create table** de la relación *departamento* garantiza que el valor de **presupuesto** no sea negativo.

Considere ahora el siguiente ejemplo:

```
create table sección
    (asignatura_id      varchar (8),
    secc_id             varchar (8),
    semestre            varchar (6),
    año                 numeric (4,0),
    edificio            varchar (15),
    número_aula         varchar (7),
    franja_horaria_id   varchar (4),
    primary key (asignatura_id, secc_id, semestre, año),
    check (semestre in (´Otoño´, ´Invierno´, ´Primavera´, ´Verano´)));
```

En este caso se utiliza la cláusula **check** para simular un tipo enumerado, especificando que *semestre* debe ser uno de ´Otoño´, ´Invierno´, ´Primavera´ o ´Verano´. Por tanto, la cláusula **check** permite restringir los atributos y los dominios de una forma potente que la mayor parte de los sistemas de tipos de lenguajes de programación no permiten.

El predicado de la cláusula **check** puede ser, según la norma de SQL, un predicado arbitrario que puede incluir una subconsulta. Sin embargo, actualmente ninguna de las bases de datos ampliamente utilizadas permite que el predicado contenga una subconsulta.

4.4.5. Integridad referencial

A menudo se desea garantizar que el valor que aparece en una relación para un conjunto dado de atributos aparezca también para un conjunto determinado de atributos en otra relación. Esta condición se denomina **integridad referencial.**

Las claves externas pueden especificarse como parte de la instrucción de SQL **create table** mediante la cláusula **foreign key.** Las declaraciones de claves externas se ilustran mediante la definición en el LDD de SQL de parte de la base de datos de la universidad, como puede verse en la Figura 4.8. La definición de la tabla *asignatura* tiene una declaración «**foreign key** (*nombre_dept*) **references** *departamento*». Esta declaración de clave externa especifica que para cada tupla de asignatura, el nombre de departamento especificado en la tupla debe existir en la relación *departamento*. Sin esta restricción, es posible que alguna asignatura especifique el nombre de un departamento inexistente.

De manera más general, sean r_1 y r_2 relaciones cuyos conjuntos de atributos son R_1 y R_2 respectivamente, con las claves primarias K_1 y K_2. Se dice que un subconjunto α de R_2 es una **clave externa** que hace referencia a K_1 de la relación r_1 si se exige que, para toda tupla t_2 de r_2, deba haber una tupla t_1 de r_1 tal que $t_1.K_1 = t_1.\alpha$.

Las exigencias de este tipo se denominan **restricciones de integridad referencial,** o **dependencias de subconjuntos.** El último término se debe a que la anterior restricción de integridad referencial puede escribirse como un requisito de que el conjunto de valores de α en r_2 debe ser un subconjunto de los valores sobre K_1 en r_1. Observe que, para que las restricciones de integridad referencial tengan sentido, α y K_1 deben ser conjuntos de atributos compatibles, es decir, o bien α debe ser igual a K_1, o bien deben contener el mismo número de atributos y los tipos de los atributos correspondientes deben ser compatibles (aquí se supone que α y K_1 están ordenados). Al contrario que las restricciones de clave externa, en general una restricción de integridad referencial no requiere que K_1 sea una clave primaria de r_1; en consecuencia, puede haber más de una tupla en r_1 que tenga los mismos valores para los atributos K_1.

De manera predeterminada, en SQL las claves externas hacen referencia a los atributos de la clave primaria de la tabla referenciada. SQL también soporta una versión de la cláusula **references** en la que se puede especificar de manera explícita una lista de atributos de la relación a la que se hace referencia. La lista de atributos especificada, no obstante, debe declararse como clave candidata de la relación a la que hace referencia, usando bien la restricción **primary key** o la restricción **unique.** Una forma más general de restricción de integridad referencial, en la que las columnas referenciadas no necesitan ser claves candidatas, no se puede especificar directamente en SQL. La norma de SQL especifica otros constructores que se pueden utilizar para estas restricciones; se describen en la Sección 4.4.7.

Se puede utilizar la siguiente forma abreviada como parte de la definición de atributos para declarar que el atributo forma una clave externa:

nombre_dept **varchar**(20) **references** *departamento*

Cuando se viola una restricción de integridad referencial, el procedimiento normal es rechazar la acción que ha causado esa violación (es decir, la transacción que lleva a cabo la acción de actualización se retrocede). Sin embargo, la cláusula **foreign key** puede especificar que si una acción de borrado o de actualización de la relación a la que hace referencia viola la restricción, entonces, en lugar de rechazar la acción, el sistema lleva a cabo los pasos necesarios para modificar la tupla de la relación que hace la referencia para que se restaure la restricción. Considere esta definición de una restricción de integridad para la relación *asignatura*:

```
create table asignatura
    (…
    foreign key (nombre_dept) references departamento
                on delete cascade
                on update cascade,
    …);
```

Debido a la cláusula **on delete cascade** asociada con la declaración de la clave externa, si el borrado de una tupla de *departamento* da lugar a que se viole la restricción de integridad referencial, el sistema no rechaza el borrado. En su lugar, el borrado «pasa en cascada» a la relación *asignatura*, borrando la tupla que hace referencia al *departamento* que se ha borrado. De manera parecida, el sistema no rechaza las actualizaciones de los campos a los que hace referencia la restricción aunque la violen; en su lugar, el sistema actualiza el campo *nombre_dept* de las tuplas referenciadas de *asignatura* también al nuevo valor. SQL permite, así mismo, que la cláusula **foreign key** especifique acciones diferentes de **cascade,** si se viola la restricción. El

campo que hace la referencia (en este caso, *nombre_departamento*) puede establecerse a *null* (empleando **set null** en lugar de **cascade**) o con el valor predeterminado para el dominio (utilizando **set default**).

Si hay una cadena de dependencias de clave externa que afecta a varias relaciones, el borrado o la actualización de un extremo de la cadena puede propagarse por toda ella. En el Ejercicio práctico 4.9 aparece un caso interesante en el que la restricción de **clave externa** de una relación hace referencia a la misma relación. Si una actualización o un borrado en cascada provocan una violación de la restricción que no pueda tratarse con otra operación en cascada, el sistema aborta la transacción. En consecuencia, todas las modificaciones provocadas por la transacción y sus acciones en cascada se deshacen.

Los valores nulos complican la semántica de las restricciones de integridad referencial en SQL. Se permite que los atributos de las claves externas sean valores nulos, siempre que no hayan sido declarados previamente como **not null.** Si todas las columnas de una clave externa son no nulas en una tupla dada, se utiliza para esa tupla la definición habitual de las restricciones de clave externa. Si alguna de las columnas de la clave externa vale *null*, la tupla se define de manera automática para que satisfaga la restricción.

Puede que esta definición no sea siempre la opción correcta, por lo que SQL ofrece también constructores que permiten modificar el comportamiento cuando hay valores nulos; estos constructores no se van a tratar aquí.

```
create table aula
  (edificio          varchar (15),
  número_aula        varchar (7),
  capacidad          numeric (4,0),
  primary key (edificio, número_aula))

create table departamento
  (nombre_dept       varchar (20),
  edificio           varchar (15),
  presupuesto        numeric (12,2) check (presupuesto > 0),
  primary key (nombre_dept))

create table asignatura
  (asignatura_id     varchar (8),
  nombre_asig        varchar (50),
  nombre_dept        varchar (20),
  créditos           numeric (2,0) check (créditos > 0),
  primary key (asignatura_id),
  foreign key (nombre_dept) references departamento)

create table profesor
  (ID                varchar (5),
  nombre             varchar (20), not null
  nombre_dept        varchar (20),
  sueldo             numeric (8,2), check (sueldo > 29000),
  primary key (ID),
  foreign key (nombre_dept) references departamento)

create table sección
  (asignatura_id     varchar (8),
  secc_id            varchar (8),
  semestre           varchar (6), check (semestre in (´Otoño´, ´Invierno´,
                                          ´Primavera´, ´Verano´),
  año                numeric (4,0), check (año > 1759 y año < 2100)
  edificio           varchar (15),
  número_aula        varchar (7),
  franja_horaria_id  varchar (4),
  primary key (asignatura_id, secc_id, semestre, año),
  foreign key (asignatura_id) references asignatura,
  foreign key (edificio, número_aula) references aula)
```

Figura 4.8. Definición de datos SQL para una parte de la base de datos de la universidad.

4.4.6. Violación de la restricción de integridad durante una transacción

Las transacciones pueden constar de varios pasos, y las restricciones de integridad pueden violarse temporalmente tras uno de los pasos, pero puede que uno posterior subsane la violación. Por ejemplo, suponga una relación *persona* con la clave primaria *nombre* y el atributo *cónyuge*, y suponga que *cónyuge* es una clave externa de *persona*. Es decir, la restricción impone que el atributo *cónyuge* debe contener un nombre que aparezca en la tabla *persona*. Suponga que se desea destacar el hecho de que Juan y María están casados entre sí mediante la inserción de dos tuplas, una para cada uno, en la relación anterior. La inserción de la primera tupla violaría la restricción de clave externa, independientemente de cuál de las dos tuplas se inserte primero. Una vez insertada la segunda tupla, la restricción de clave externa vuelve a cumplirse.

Para tratar estas situaciones, la norma de SQL permite que se añada la cláusula **initially deferred** a la especificación de la restricción; la restricción, en este caso, se comprueba al final de la transacción y no en los pasos intermedios. Las restricciones pueden especificarse de manera alternativa como **deferrable,** que significa que, de manera predeterminada, se comprueba inmediatamente, pero puede diferirse si se desea. Para las restricciones declaradas como diferibles, la ejecución de la instrucción **set constraints** *lista-restricciones* **deferred** como parte de una transacción hace que se difiera la comprobación de las restricciones especificadas hasta el final de esa transacción.

No obstante, hay que ser consciente de que el comportamiento predeterminado es comprobar las restricciones de manera inmediata, y de que muchas implementaciones no soportan la comprobación diferida de las restricciones.

Se puede tratar el problema anterior de otra forma, si el atributo *cónyuge* se puede establecer a *null*. Se establecen ambos atributos a *null* cuando se insertan las tuplas de Juan y María y ambas se actualizan más tarde. Sin embargo, esta técnica requiere un mayor esfuerzo de programación y no funciona si los atributos no pueden valer *null*.

4.4.7. Condiciones de comprobación complejas y asertos

La norma SQL soporta construcciones adicionales para especificar las restricciones de integridad que se describen en esta sección. Sin embargo, debería ser consciente que estas construcciones no las soportan la mayoría de los sistemas de bases de datos.

Como se define en la norma de SQL, el predicado de la cláusula **check** puede ser un predicado arbitrario, que puede incluir una subconsulta. Si una implementación de bases de datos soporta subconsultas en la cláusula **check,** se podría especificar la siguiente restricción de integridad referencial en la relación *sección*:

check (*franja_horaria_id* **in** (**select** *franja_horaria_id* **from** *franja_horaria*))

La condición de **check** verifica que el *franja_horaria_id* de las tuplas en la relación *sección* es realmente el identificador de una franja horaria en la relación *franja_horaria*. Por tanto, la condición se tiene que comprobar, no solo cuando se inserta o modifica una tupla en *sección*, sino también cuando la relación *franja_horaria* cambia (en este caso, cuando se borra o modifica una tupla de la relación *franja_horaria*).

Otra restricción natural en el esquema de la universidad sería requerir que todas las secciones tuviesen al menos un profesor que enseñase en la misma. En un intento de forzar este hecho se puede intentar declarar que los atributos (*asignatura_id*, *secc_id*, *semestre*, *año*) de la relación *sección* formen una clave externa que referencie a los atributos correspondientes de la relación *enseña*. Desafortunadamente, estos atributos no forman una clave candidata de la relación

enseña. Se podría utilizar una restricción de comprobación similar a la indicada para el atributo *franja_horaria* si el sistema de bases de datos soportase las restricciones de comprobación con subconsultas.

Las condiciones **check** complejas pueden ser útiles cuando se desea asegurar la integridad de los datos, pero puede ser costoso comprobarlo. Por ejemplo, el predicado de la cláusula **check** no solo tiene que evaluarse cuando se realiza una modificación en la relación *sección*, sino que puede que tenga que comprobarse si se realiza una modificación en la relación *franja_horaria*, ya que esta relación se referencia en la subconsulta.

Un **aserto** es un predicado que expresa una condición que la base de datos debe satisfacer siempre. Las restricciones de dominio y las de integridad referencial son formas especiales de asertos. Se ha prestado una atención especial a estos tipos de asertos porque se pueden verificar con facilidad y pueden afectar a una gran variedad de aplicaciones de bases de datos. Sin embargo, hay muchas restricciones que no se pueden expresar utilizando únicamente estas formas especiales. Dos ejemplos de estas restricciones son:

- Para las tuplas de la relación *estudiante*, el valor del atributo *tot_créditos* debe ser igual a la suma de los créditos de las asignaturas que el estudiante ha aprobado.
- Un profesor no puede enseñar en dos aulas en un semestre en la misma franja horaria.[1]

  En SQL los asertos adoptan la forma:

  create assertion <nombre aserto> **check** <predicado>;

En la Figura 4.9 se muestra cómo se pueden escribir estos dos ejemplos de restricciones en SQL. Dado que SQL no proporciona ningún mecanismo «para todo X, $P(X)$» (donde P es un predicado), hay que implementarlo utilizando su equivalente «no existe X tal que no $P(X)$», que se puede expresar en SQL.

Se deja la especificación de la segunda restricción como ejercicio.

Cuando se crea un aserto, el sistema comprueba su validez. Si el aserto es válido, solo se permiten las modificaciones posteriores de la base de datos que no hagan que se viole el aserto. Esta comprobación puede introducir una sobrecarga importante si se han realizado asertos complejos. Por tanto, los asertos deben utilizarse con mucha cautela. La elevada sobrecarga debida a la comprobación y al mantenimiento de los asertos ha llevado a algunos desarrolladores de sistemas a soslayar el soporte para los asertos generales, o bien a proporcionar formas especializadas de aserto que resultan más sencillas de comprobar.

En la actualidad, ninguno de los sistemas de bases de datos soporta las subconsultas en el predicado de la cláusula **check,** ni las construcciones **assertion.** Sin embargo, se puede implementar una funcionalidad equivalente usando disparadores, que se describen en la Sección 5.3, si dispone de ellos el sistema de bases de datos. En la Sección 5.3 también se describe cómo se puede implementar la restricción de integridad referencial sobre *franja_horaria_id* utilizando disparadores.

```
create assertion créditos_obtenidos_restricción check
(not exists (select ID
             from estudiante
             where tot_créditos <> (select sum(créditos)
                from enseña natural join asignatura
                where estudiante.ID = enseña.ID
                      and nota is not null and nota<> ´F´)))
```

Figura 4.9. Ejemplo de aserto.

1 Se supone que las clases no se transmiten de forma remota a una segunda aula. Una restricción alternativa que especifica que «un profesor no puede enseñar dos asignaturas en un mismo semestre en la misma franja horaria» puede que a veces no se cumpla si las asignaturas se cruzan, es decir, si a la misma asignatura se le dan dos identificadores y dos títulos distintos.

4.5. Tipos de datos y esquemas de SQL

En el Capítulo 3 se vieron varios tipos de datos predefinidos soportados por SQL, como los tipos enteros, los reales y los de carácter. Hay otros tipos de datos predefinidos en SQL, que se describirán a continuación. También se describirá la manera de crear en SQL tipos sencillos definidos por los usuarios.

4.5.1. Tipos fecha y hora en SQL

Además de los tipos de datos básicos que se presentaron en la Sección 3.2, la norma SQL soporta otros tipos de datos relacionados con fecha y hora:

- **date.** Fecha del calendario que contiene el año (con cuatro dígitos), el mes y el día del mes.
- **time.** La hora del día, en horas, minutos y segundos. Se puede usar una variante, time(p), para especificar el número de cifras decimales para los segundos (el valor predeterminado es 0). También es posible almacenar la información del huso horario junto a la hora especificando **time with timezone.**
- **timestamp.** Una combinación de **date** y **time.** Se puede usar una variante, timestamp(p), para especificar el número de cifras decimales para los segundos (el valor predeterminado es seis). También se almacena información sobre el huso horario si se especifica **with timezone.**

Los valores de fecha y hora se pueden especificar de esta manera:

date ´2001-04-25´
time ´09:30:00´
timestamp ´2001-04-25 10:29:01.45´

La fecha se debe especificar en el formato de año seguido del mes y del día, tal y como se muestra. El campo segundos de **time** y **timestamp** puede tener una parte decimal, como puede verse en el ejemplo anterior.

Se pueden usar expresiones de la forma **cast** e **as** t para convertir una cadena de caracteres (o una expresión de tipo cadena de caracteres) e al tipo t, donde t es de tipo **date**, **time** o **timestamp.** La cadena de caracteres debe tener el formato adecuado, como se indicó al comienzo de este párrafo. Si fuese necesario, la información sobre el huso horario puede deducirse de la configuración del sistema.

Para extraer campos concretos de un valor d de **date** o de **time** se puede utilizar **extract**(*campo* **from** d), donde *campo* puede ser **year, month, day, hour, minute** o **second.** La información de hora sobre el huso horario puede obtenerse mediante **timezone_hour** y **timezone_minute.**

SQL también define varias funciones útiles para obtener la fecha y la hora actuales. Por ejemplo, **current_date** devuelve la fecha actual, **current_time** devuelve la hora actual (con su huso horario) y **localtime** devuelve la hora local actual (sin huso horario). Las marcas de tiempo (fecha y hora) se obtienen con **current_timestamp** (con huso horario) y **localtimestamp** (fecha y hora locales sin huso horario).

SQL permite realizar operaciones de comparación sobre todos los tipos de datos que se han mencionado, así como operaciones aritméticas y de comparación sobre los diferentes tipos de datos numéricos. SQL también proporciona un tipo de datos denominado **interval** y permite realizar cálculos basados en fechas, horas e intervalos. Por ejemplo, si x e y son del tipo **date,** entonces $x - y$ es un intervalo cuyo valor es el número de días desde la fecha x hasta la fecha y. De forma análoga, al sumar o restar un intervalo a una fecha o a una hora se obtiene como resultado otra fecha u hora, respectivamente.

4.5.2. Valores predeterminados

SQL permite especificar un valor predeterminado para un atributo, como se muestra en la siguiente sentencia **create table:**

create table *estudiante*
(*ID* **varchar** (5),
nombre **varchar** (20) **not null**,
nombre_dept **varchar** (20),
tot_créditos **numeric** (3,0) **default** 0,
primary key (*ID*));

El valor predeterminado del atributo *tot_créditos* se declara como 0. En consecuencia, cuando se inserta una tupla en la relación *estudiante*, si no se proporciona un valor para el atributo *tot_créditos* su valor se pone a 0. La siguiente sentencia muestra cómo se puede omitir el valor del atributo *tot_créditos*:

insert into *estudiante*(*ID*, *nombre*, *nombre_dept*)
values (´12789´, ´Newman´, ´Informática´);

4.5.3. Creación de índices

Muchas consultas solo hacen referencia a una pequeña parte de los registros de un archivo. Por ejemplo, las consultas «encontrar todos los profesores del departamento de Física», o «encontrar el valor de *tot_créditos* del estudiante con *ID* 22201» solo hacen referencia a una fracción del total de registros. Resulta ineficiente que el sistema lea todos los registros y compruebe el campo *ID* para ver si es el *ID* «32556» o si el campo *edificio* es «Física».

Un **índice** sobre un atributo de una relación es una estructura de datos que permite al sistema de bases de datos encontrar de forma eficiente aquellas tuplas de la relación que tienen un determinado valor del atributo, sin tener que explorar todas las tuplas de la relación. Por ejemplo, si se crea un índice para el atributo *ID* de la relación *estudiante*, el sistema de la base de datos puede encontrar el registro con cualquier valor de *ID*, como 22201, o 44553, directamente, sin leer todas las tuplas de la relación *estudiante*. También se puede crear un índice de una lista de atributos, por ejemplo sobre los atributos *nombre*, *nombre_dept* o *estudiante*.

Más adelante, en el Capítulo 11, se trata cómo se implementan realmente los índices, incluyendo un tipo particular de índices muy utilizados denominados índices de árboles B^+.

Aunque el lenguaje SQL no define formalmente ninguna sintaxis para la creación de índices, muchas bases de datos soportan la creación de índices con la siguiente sintaxis:

create index *estudianteID_índice* **on** *estudiante*(*ID*);

La sentencia anterior crea un índice denominado *estudianteID_índice* sobre el atributo *ID* de la relación *estudiante*.

Cuando un usuario envía una consulta de SQL que se puede aprovechar de dicho índice, el procesador de consultas de SQL utiliza automáticamente el índice. Por ejemplo, dada una consulta de SQL que selecciona la tupla del *estudiante* con *ID* 22201, el procesador de consultas de SQL utilizaría el índice definido anteriormente *estudianteID_índice* para encontrar la tupla necesaria sin leer la relación completa.

4.5.4. Tipos de datos para objetos grandes

Muchas aplicaciones de bases de datos de última generación necesitan almacenar atributos que pueden ser grandes (del orden de muchos kilobytes), como pueden ser las fotografías de personas, o muy grande (del orden de muchos megabytes o, incluso, gigabytes), como las imágenes médicas de alta resolución o los fragmentos de vídeo. SQL, por tanto, ofrece nuevos tipos de datos para objetos de gran tamaño para los datos de caracteres **(clob)** y para los datos binarios **(blob).** Las letras «lob» de estos tipos de datos significan «objeto grande» (large object). Por ejemplo, se pueden declarar los atributos:

revista **clob**(10KB)
imagen **blob**(10MB)
película **blob**(2GB)

Para tuplas resultado que contienen objetos muy grandes (de varios megabytes a gigabytes), resulta poco eficiente o poco práctico recuperar en la memoria objetos de gran tamaño de tipo entero. En su lugar, las aplicaciones suelen utilizar las consultas SQL para recuperar «localizadores» de los objetos de gran tamaño y luego emplean el localizador para manipular el objeto desde el lenguaje anfitrión en que está escrita la aplicación. Por ejemplo, la interfaz de programación de aplicaciones JDBC (que se describe en la Sección 5.1.1) permite recuperar un localizador en lugar de todo el objeto de gran tamaño; luego se puede emplear el localizador para recuperar este objeto en fragmentos más pequeños, en vez de recuperarlo todo de una vez, de manera parecida a como se leen los datos de los archivos del sistema operativo mediante llamadas a funciones de lectura.

4.5.5. Tipos definidos por los usuarios

SQL soporta dos formas de tipos de datos definidos por los usuarios. La primera forma, que se tratará a continuación, se denomina **alias de tipos** (*distinct types*). La otra forma, denominada **tipos de datos estructurados,** permite la creación de tipos de datos complejos, que pueden anidar estructuras de registro, arrays y multiconjuntos. En este capítulo no se tratan los tipos de datos complejos, pero se describen más adelante, en el Capítulo 22.

Varios atributos pueden ser del mismo tipo de datos. Por ejemplo, los atributos *nombre* para el nombre de un estudiante y para el de un profesor pueden tener el mismo dominio: el conjunto de todos los nombres de personas. No obstante, los dominios de *presupuesto* y *nombre_dept*, ciertamente, deben ser diferentes. Quizás resulte menos evidente si *nombre* y *nombre_dept* deben tener el mismo dominio. En el nivel de implementación, tanto los nombres de los profesores como los de los departamentos son cadenas de caracteres. Sin embargo, normalmente no se considera que la consulta «encontrar todos los profesores que tengan el mismo nombre que algún departamento» sea una consulta con sentido. Por tanto, si se considera la base de datos desde el nivel conceptual, en vez de hacerlo desde el nivel físico, *nombre* y *nombre_dept* deben tener dominios distintos.

A nivel práctico, asignar el nombre de un profesor a un departamento probablemente sea un error de programación; de la misma forma, comparar directamente un valor monetario expresado en euros con otro valor monetario expresado en libras también es, seguramente, un error de programación. Un buen sistema de tipos de datos debe poder detectar este tipo de asignaciones o comparaciones. Para dar soporte a estas comprobaciones, SQL aporta el concepto de **alias de tipos.**

La cláusula **create type** puede utilizarse para definir tipos de datos nuevos. Por ejemplo, las sentencias:

create type *Euros* **as numeric**(12,2) **final**;
create type *Libras* **as numeric**(12,2) **final**;

declaran los tipos de datos definidos por los usuarios *Euros* y *Libras* como números decimales con un total de doce dígitos, dos de los cuales se encuentran tras la coma decimal. (La palabra clave **final** no resulta realmente significativa en este contexto, pero la norma de SQL:1999 la exige por motivos que no vienen al caso; algunas implementaciones permiten omitir la palabra clave **final**).

Los tipos recién creados pueden utilizarse, por ejemplo, como tipos de los atributos de las relaciones. Por ejemplo, se puede declarar la tabla *departamento* como:

```
create table departamento
  (nombre_dept varchar (20),
  edificio     varchar (15),
  presupuesto  Euros);
```

Cualquier intento de asignar un valor de tipo *Euros* a una variable de tipo *Libras* daría como resultado un error de compilación, aunque ambos sean del mismo tipo numérico. Una asignación de ese tipo probablemente se deba a un error de programación, si el programador ha olvidado las diferencias entre divisas. La declaración de tipos distintos para divisas diferentes ayuda a detectar esos errores.

Como consecuencia del control riguroso de los tipos de datos, la expresión (*department.budget*+20) no se aceptaría, ya que el atributo y la constante entera 20 tienen diferentes tipos de datos. Los valores de un tipo de datos pueden convertirse (*cast*) a otro dominio como se muestra a continuación:

```
cast (departamento.presupuesto to numeric(12,2))
```

Se podría realizar la suma sobre el tipo **numeric,** pero para volver a guardar el resultado en un atributo del tipo *Euros* habría que emplear otra expresión *cast* para volver a convertir el tipo de datos a *Euros*.

SQL también ofrece las cláusulas **drop type** y **alter type** para eliminar o modificar los tipos de datos que se han creado anteriormente.

Antes incluso de la incorporación a SQL de los tipos de datos definidos por los usuarios (en SQL:1999), SQL tenía el concepto parecido, pero sutilmente diferente, de **dominio** (introducido en SQL-92), capaz de añadir restricciones de integridad a un tipo subyacente. Por ejemplo, se podría definir el tipo de dominio *DEuros* de la manera siguiente:

```
create domain DEuros as numeric(12,2) not null;
```

El tipo de dominio *DEuros* puede utilizarse como tipo de atributo, igual que se ha utilizado el tipo de datos *Euros*. Sin embargo, hay dos diferencias significativas entre los tipos de datos y los dominios:

1. Sobre los dominios se pueden especificar restricciones, como **not null,** y valores predeterminados para las variables del tipo de dominio, pero no se pueden especificar restricciones ni valores predeterminados sobre los tipos de datos definidos por los usuarios. Los tipos de datos definidos por los usuarios están diseñados no solo para utilizarlos con el fin de especificar los tipos de datos de los atributos, sino también en extensiones procedimentales de SQL en las que puede que no sea posible aplicar restricciones.
2. Los dominios no tienen tipos estrictos. En consecuencia, los valores de un tipo de dominio pueden asignarse a los de otro tipo de dominio, siempre y cuando los tipos subyacentes sean compatibles.

Cuando se aplica a un dominio, la cláusula **check** permite al diseñador del esquema especificar un predicado que todos los atributos declarados deben satisfacer en este dominio. Por ejemplo, con una cláusula **check** se puede asegurar que el dominio sueldo de un profesor solo admite valores superiores a un valor dado:

```
create domain SueldoAnual numeric(8,2)
        constraint valor_sueldo_test
                check(value >= 29000.00);
```

El dominio *SueldoAnual* tiene una restricción que asegura que el sueldo anual es mayor o igual a 29,000.00 €. La cláusula **constraint** *valor_sueldo_test* es opcional, y se usa para dar el nombre *valor_sueldo_test* a la restricción. El nombre lo utiliza el sistema para indicar qué restricción no cumple una determinada actualización.

Otro ejemplo: un dominio se puede restringir a que contenga solo un determinado conjunto de valores usando la cláusula **in:**

```
create domain nivel_estudios varchar(10)
        constraint nivel_estudios_test
                check (value in ('Bachiller', 'Máster',
                        or 'Doctorado'));
```

SOPORTE PARA TIPOS Y DOMINIOS EN LAS IMPLEMENTACIONES DE BASES DE DATOS

Aunque las construcciones **create type** y **create domain** que se describen en esta sección forman parte de la norma SQL, los formatos no los soportan totalmente la mayoría de implementaciones de las bases de datos. PorstgreSQL soporta la construcción **create domain,** pero su construcción **create type** tiene una sintaxis e interpretación diferentes.

DB2 de IBM soporta una versión de **create type** que usa la sintaxis **create distinct type,** pero no soporta **create domain.** SQL Server de Microsoft implementa una versión de la construcción **create type** que soporta restricciones de dominio, similar a la construcción **create domain** de SQL.

Oracle no soporta ninguna de las construcciones descritas. Sin embargo, SQL también define un sistema de tipos orientado a objetos más complejo, que se estudiará más adelante en el Capítulo 22. Oracle, DB2 de IBM, PostgreSQL y SQL Server soportan los sistemas de tipos orientados a objetos utilizando distintas formas de la construcción **create type.**

4.5.6. Extensiones a la creación de tablas

A menudo, las aplicaciones requieren la creación de tablas que tienen el mismo esquema que una ya existente. SQL proporciona una extensión **create table like** para esta tarea:

```
create table temp_profesor like profesor;
```

La sentencia anterior crea una nueva tabla *temp_profesor* que tiene el mismo esquema que *profesor*.

Cuando se escribe una consulta compleja, a menudo suele ser útil guardar el resultado de la consulta como una nueva tabla; la tabla suele ser temporal. Se necesitan dos sentencias, una para crear la tabla (con las columnas apropiadas) y una segunda para insertar los resultados en ella. SQL:2003 proporciona una técnica más simple para crear una tabla que contenga los resultados de una consulta. Por ejemplo, la siguiente sentencia crea una tabla *t1* que contiene los resultados de la consulta:

```
create table t1 as
    (select *
    from profesor
    where nombre_dept = 'Música')
with data;
```

De forma predeterminada, los nombres y los tipos de datos de las columnas se infieren del resultado de la consulta. Se pueden dar nombres explícitamente a las columnas indicando la relación de nombres de columna tras el nombre de la relación.

Como se define en la norma SQL:2003, si se omite la cláusula **with data,** la tabla se crea pero no se rellena con datos. Sin embargo, muchas implantaciones rellenan la tabla con datos de forma predeterminada incluso aunque se omita la cláusula **with data.**

Tenga en cuenta que muchas implementaciones soportan la funcionalidad de **create table… like** y **create table… as** utilizando una sintaxis diferente; consulte los manuales del sistema respectivos para obtener más información.

La sentencia anterior **create table… as** se parece mucho a la sentencia **create view** y ambas se definen mediante consultas. La diferencia más importante es que el contenido de la tabla se inserta cuando se crea, mientras que el contenido de una vista siempre refleja el resultado actualizado de la consulta.

4.5.7. Esquemas, catálogos y entornos

Para comprender la razón de que existan esquemas y catálogos, considere la manera en que se dan nombres a los archivos en los sistemas de archivos. Los primeros sistemas de archivos eran «planos», es decir, todos los archivos se almacenaban en un único directorio. Los sistemas de archivos actuales, evidentemente, tienen una estructura de directorios y los archivos se almacenan en diferentes subdirectorios (o su sinónimo carpeta). Para dar un nombre único a un archivo hay que especificar su nombre de ruta completo, por ejemplo, /usuarios/avi/libro-bd/capítulo3.tex.

Al igual que los primeros sistemas de archivos, los primeros sistemas de bases de datos tenían también un único espacio de nombres para todas las relaciones. Los usuarios tenían que coordinarse para asegurarse de que no intentaban emplear el mismo nombre para relaciones diferentes. Los sistemas de bases de datos contemporáneos ofrecen una jerarquía de tres niveles para los nombres de las relaciones. El nivel superior de la jerarquía consta de **catálogos,** cada uno de los cuales puede contener **esquemas.** Los objetos de SQL, como las relaciones y las vistas, están contenidos en **esquemas.** (Algunas implementaciones de las bases de datos utilizan el término «base de datos» en lugar de «catálogo»).

Para llevar a cabo cualquier acción sobre la base de datos los usuarios (o los programas) primero deben *conectarse* a ella. El usuario debe proporcionar el nombre de usuario y, generalmente, una contraseña secreta para que se compruebe su identidad. Cada usuario tiene un catálogo y un esquema predeterminados, y esa combinación es única para cada usuario. Cuando un usuario se conecta a un sistema de bases de datos, se configuran para la conexión el catálogo y el esquema predeterminados; esto equivale a la conversión del directorio actual en el directorio de inicio del usuario cuando este inicia sesión en un sistema operativo.

Para identificar de manera única cada relación hay que utilizar un nombre con tres partes. Por ejemplo:

catálogo5.univ_esquema.asignatura

Se puede omitir el componente del catálogo, en cuyo caso la parte del nombre correspondiente al catálogo se considera que es el catálogo predeterminado de la conexión. Por tanto, si *catálogo5* es el catálogo predeterminado, se puede utilizar *univ_esquema.asignatura* para identificar de manera única la misma relación.

Si un usuario quiere acceder a una relación que existe en un esquema distinto del esquema predeterminado para dicho usuario, se debe especificar el nombre del esquema. Sin embargo, si una relación se encuentra en el esquema predeterminado para un usuario en concreto, entonces incluso se puede omitir el nombre del esquema. Por tanto, se puede emplear únicamente *asignatura* si el catálogo predeterminado es *catálogo5* y el esquema predeterminado es *univ_esquema*.

Cuando se dispone de varios catálogos y de varios esquemas, diferentes aplicaciones y usuarios pueden trabajar de manera independiente sin preocuparse de posibles coincidencias de nombres. Además, pueden ejecutarse varias versiones de la misma aplicación —una versión de producción, otras versiones de prueba— sobre el mismo sistema de bases de datos.

El catálogo y el esquema predeterminados forman parte de un **entorno** de SQL que se configura para cada conexión. El entorno contiene, además, el identificador del usuario (también denominado *identificador de autorización*). Todas las sentencias habituales de SQL, incluidas las sentencias de LDD y de LMD, operan en el contexto de un esquema.

Se pueden crear y descartar esquemas mediante las sentencias **create schema** y **drop schema.** En la mayoría de los sistemas de bases de datos, los esquemas también se crean automáticamente cuando se crean las cuentas de usuario, en las que el nombre del esquema se establece al nombre de la cuenta del usuario. El esquema se crea bien en un catálogo predeterminado o en un catálogo especificado al crear la cuenta de usuario. El nuevo esquema creado se convierte en el esquema predeterminado para la cuenta de usuario.

La creación y la eliminación de catálogos dependen de cada implementación y no forman parte de la norma de SQL.

4.6. Autorización

Se pueden asignar a los usuarios varios tipos de autorización para diferentes partes de la base de datos. Las autorizaciones sobre datos incluyen:

- La autorización de lectura de datos.
- La autorización de inserción de nuevos datos.
- La autorización de actualización de datos.
- La autorización de borrado de datos.

Cada uno de estos tipos de autorización se denomina **privilegio.** Se puede autorizar a cada usuario para que tenga todos estos tipos de privilegios, ninguno de ellos o una combinación de los mismos sobre partes concretas de la base de datos, como pueden ser una relación o una vista.

Cuando un usuario envía una consulta o una actualización, la implementación de SQL primero comprueba si la consulta o actualización están autorizadas, de acuerdo con las autorizaciones que se hayan concedido al usuario. Si la consulta o actualización no están autorizadas, se rechazan.

Además de la autorización sobre los datos, los usuarios también reciben concesiones de autorización sobre los esquemas de la base de datos, permitiéndoles, por ejemplo, crear, modificar o eliminar relaciones. Un usuario que disponga de alguna forma de autorización puede que pase (o conceda) esta autorización a otros usuarios, o pierda (revoque) una autorización que se le concedió con anterioridad. En esta sección se verá cómo se pueden especificar estas autorizaciones en SQL.

El último elemento de autoridad es el que se da al administrador de la base de datos. El administrador puede autorizar nuevos usuarios, reestructurar la base de datos y mucho más. Esta forma de autorización es análoga a la de un superusuario, un administrador o un operador de un sistema operativo.

4.6.1. Concesión y revocación de privilegios

La norma de SQL incluye los privilegios **select, insert, update** y **delete.** El privilegio **all privileges** se puede utilizar como forma abreviada de todos los privilegios que se pueden conceder. El usuario que crea una relación nueva recibe de manera automática todos los privilegios sobre esa relación.

El lenguaje de definición de datos de SQL incluye comandos para conceder y revocar privilegios. La instrucción **grant** se utiliza para conceder autorizaciones. La forma básica de esta sentencia es:

grant <lista de privilegios>
on <nombre de relación o de vista>
to <lista de usuarios o de roles>;

La *lista de privilegios* permite conceder varios privilegios en un solo comando. La noción de roles se trata más adelante, en la Sección 4.6.2.

La autorización **select** sobre una relación es necesaria para leer las tuplas de una relación. La siguiente sentencia **grant** garantiza a los usuarios de la base de datos Amit y Satoshi la autorización **select** sobre la relación *departamento*:

grant select on *departamento* **to** *Amit*, *Satoshi*;

Esto permite a dichos usuarios ejecutar consultas sobre la relación *departamento*.

La autorización **update** sobre una relación permite a los usuarios actualizar cualquier tupla de la relación. La autorización **update** puede concederse sobre todos los atributos de la relación o solo sobre algunos. Si se incluye la autorización **update** en una sentencia **grant,** la lista de atributos sobre los que se garantiza la autorización **update** aparece opcionalmente entre paréntesis justo después de la palabra clave **update.** Si se omite la lista de atributos, se garantiza el privilegio de actualización sobre todos los atributos de la relación.

La siguiente sentencia **grant** concede a los usuarios Amit y Satoshi la autorización de actualizar sobre el atributo *presupuesto* de la relación *departamento*:

grant update (*presupuesto*) **on** *departamento*
to *Amit*, *Satoshi*;

La autorización **insert** sobre una relación permite a los usuarios insertar tuplas en la relación. El privilegio **insert** también puede especificar una lista de atributos; cualquier inserción en la relación debe especificar solo esos atributos y el sistema asigna al resto de los atributos valores predeterminados (si hay alguno definido para ellos) o los define como nulos.

La autorización **delete** sobre una relación permite a los usuarios borrar tuplas de una relación.

El nombre de usuario **public** hace referencia a todos los usuarios actuales y futuros del sistema. Por tanto, los privilegios concedidos a **public** se garantizan de manera implícita a todos los usuarios actuales y futuros.

De manera predeterminada, el usuario o rol al que se le concede un privilegio no está autorizado a concedérselo a otro usuario o rol. SQL permite que la concesión de privilegios especifique que el destinatario puede concedérselo, a su vez, a otro usuario. Esta característica se describe con más detalle en la Sección 4.6.5.

Es importante tener en cuenta que el mecanismo de autorización de SQL concede privilegios sobre una relación completa o sobre determinados atributos de una relación. Sin embargo, no permite la autorización sobre tuplas concretas de una relación.

Para revocar una autorización se emplea la sentencia **revoke.** Su forma es casi idéntica a la de **grant:**

revoke <lista de privilegios>
on <nombre de la relación o nombre de la vista>
from <lista de usuarios o de roles>;

Por tanto, para revisar los privilegios que se han concedido anteriormente, se escribe:

revoke select on *departamento* **from** *Amit*, *Satoshi*;
revoke update (*presupuesto*) **on** *departamento*
from *Amit*, *Satoshi*;

La revocación de privilegios resulta más compleja si el usuario al que se le revocan los privilegios se los ha concedido a otros usuarios. Se volverá a este problema en la Sección 4.6.5.

4.6.2. Roles

Considere en el mundo real los roles de distintas personas en una universidad. Todos los profesores deben tener los mismos tipos de autorizaciones sobre el mismo conjunto de relaciones. Cuando llega un nuevo profesor, se le conceden todas esas autorizaciones de forma individual.

El enfoque más adecuado consistiría en especificar las autorizaciones que deberían tener todos los profesores e identificar por separado qué usuarios de la base de datos son profesores. El sistema puede usar estos dos elementos de información para determinar las autorizaciones de los profesores. Cuando se contrata a un nuevo profesor, se le debe asignar un nuevo identificador, e identificarlo como profesor. Ya no es necesario indicar permisos individuales a los profesores.

La noción de **roles** captura este concepto. Se crea un conjunto de roles en la base de datos. Las autorizaciones se conceden a los roles de la misma forma que se conceden a los usuarios individuales. A los usuarios de la base de datos se les concede un conjunto de roles (que puede estar vacío) que los usuarios pueden realizar.

En la base de datos de la universidad, ejemplos de roles pueden incluir *profesor*, *profesor_ayudante*, *estudiante*, *decano* y *jefe_departamento*.

Una alternativa menos conveniente sería crear un identificador de usuario de *profesor* y permitir a todos los profesores que utilizasen ese identificador para conectarse a la base de datos. El problema con este enfoque es que no sería posible identificar exactamente qué profesor llevó a cabo cierta actualización de la base de datos, lo que conlleva riesgos de seguridad. El uso de roles tiene la ventaja de requerir que los usuarios se conecten a la base de datos con su propio identificador.

Todas las autorizaciones que se concedan a un usuario se pueden conceder a un rol. Los roles se conceden a los usuarios igual que las autorizaciones.

Los roles se pueden crear en SQL de la siguiente forma:

create role *profesor*;

Se pueden conceder privilegios a los roles igual que a los usuarios, como se muestra en la siguiente sentencia:

grant select on *matricula*
to *profesor*;

Se pueden conceder roles a los usuarios, así como a otros roles, como se muestra a continuación:

grant *decano* **to** Amit;
create role *decano*;
grant *profesor* **to** *decano*;
grant *decano* **to** Satoshi;

Por tanto, los privilegios de un usuario o un rol constan de:

- Todos los privilegios que se concedan directamente al usuario o al rol.
- Todos los privilegios que se concedan a los roles que se hayan concedido al usuario o al rol.

Tenga en cuenta que puede haber una cadena de roles; por ejemplo, el rol de *profesor_ayudante* se puede conceder a todos los *profesores*. Y el rol de *profesor* se concede a todos los *decanos*. Por tanto, el rol de *decano* hereda todos los privilegios concedidos a los roles de *profesor* y de *profesor_ayudante*, además de los privilegios que se conceden directamente a *decano*.

Cuando un usuario se registra en el sistema de bases de datos, las acciones que ejecuta el usuario durante la sesión tienen los privilegios concedidos al usuario, así como todos los privilegios concedidos a los roles que se le han concedido (directa o indi-

rectamente a través de otros roles). Por tanto, si el usuario Amit tiene concedido el rol de *decano*, el usuario Amit tiene todos los privilegios que se conceden a Amit, así como los privilegios que se conceden a *decano*, más los privilegios concedidos a *profesor* y a *profesor_ayudante* si, como anteriormente, estos roles se conceden (directa o indirectamente) al rol *decano*.

Hay que tener en cuenta que el concepto de autorización por rol no es específico de SQL y se usa para el control de una gran variedad de aplicaciones compartidas.

4.6.3. Autorización sobre vistas

En nuestro ejemplo de la universidad, considere a los miembros de la dirección que necesitan conocer los sueldos de todos los empleados de un determinado departamento, por ejemplo, el de Geología. Estos miembros de la dirección no tienen autorización para ver la información sobre empleados de otros departamentos. Por tanto, a los miembros de la dirección se les debe denegar el acceso a la relación *profesor*. Pero si tienen que tener acceso a la información del departamento de Geología, se les podría conceder el acceso a una vista que se llamará *profesor_geología*, que consta de todas aquellas tuplas de *profesor* pertenecientes al departamento de Geología. Esta vista se puede definir en SQL como:

create view *profesor_geología* **as**
(**select** *
from *profesor*
where *nombre_dept* = ´Geología´);

Suponga que los miembros de la dirección envían la siguiente consulta de SQL:

select *
from *profesor_geología*;

Claramente, a los miembros de la dirección se les autoriza a consultar este resultado. Sin embargo, cuando el procesador de consultas traslada esta consulta sobre las relaciones reales de la base de datos, genera una consulta sobre *profesor*. Por tanto, el sistema debe comprobar la autorización sobre la consulta antes de empezar el procesamiento de la misma.

Un usuario que crea una vista no necesariamente recibe todos los privilegios sobre dicha vista, solo recibe aquellos privilegios que no proporcionan autorización adicional más allá de los que ya tiene. Por ejemplo, un usuario que crea una vista no puede obtener una autorización de **update** sobre esta sin tener la autorización de **update** sobre las relaciones que se utilizan para definir la vista. Si un usuario crea una vista sobre la que no se le puede conceder ninguna autorización, el sistema denegará la petición de creación de la vista. En nuestro ejemplo del *profesor_geología*, el creador de la vista debe tener autorización **select** sobre la relación *profesor*.

Como se tratará más adelante, en la Sección 5.2, SQL soporta la creación de funciones y procedimientos, que pueden a su vez contener consultas y actualizaciones. El privilegio **execute** se puede conceder a una función o a un procedimiento, permitiendo a un usuario ejecutar la función o procedimiento. De forma predeterminada, como con las vistas, las funciones y procedimientos tienen todos los privilegios que tenía el creador de los mismos. En efecto, la función o procedimiento se ejecuta como si lo invocase el usuario que los creó.

Aunque este comportamiento resulta apropiado en muchas situaciones, no es siempre el apropiado. Desde SQL:2003 si la definición de la función tiene una cláusula extra **sql security invoker,** entonces se ejecuta con los privilegios del usuario que invoca la función, en lugar de con los privilegios de quien **define** la función. Esto permite la creación de bibliotecas de funciones que se ejecutan con los mismos privilegios que quien las invoca.

4.6.4. Autorizaciones sobre esquemas

La norma de SQL especifica un mecanismo de autorización para el esquema de la base de datos: solo el propietario del esquema puede realizar modificaciones en el mismo, como crear o borrar relaciones, añadir y eliminar atributos y añadir o eliminar índices.

Sin embargo, SQL incluye un privilegio de referenciar **(references),** que permite que un usuario declare claves externas cuando crea relaciones. El privilegio se concede sobre atributos concretos, de la misma forma que el privilegio **update.** La siguiente sentencia **grant** permite a Mariano crear relaciones que hacen referencia a la clave *nombre_dept* de la relación *departamento* como clave externa:

grant references (*nombre_dept*) **on** *departamento* **to** Mariano;

Inicialmente, puede parecer que no existe ninguna razón para evitar que los usuarios creen claves externas que hagan referencia a otra relación. Sin embargo, recuerde que las restricciones de clave externa pueden evitar la realización de operaciones de borrado o actualización sobre la relación referenciada. Suponga que Mariano crea una clave externa en una relación *r* referenciando al atributo *nombre_dept* de la relación *departamento* e inserta una tupla en *r* perteneciente al departamento de Geología. Ya no será posible borrar el departamento de Geología de la relación *departamento* sin modificar la relación *r*. Por tanto, la definición de una clave externa por parte de Mariano restringe actividades futuras por parte de otros usuarios; entonces, existe la necesidad de estos privilegios **references.**

Continuando con el ejemplo de la relación *departamento*, el privilegio de referencia sobre *departamento* también es necesario para crear restricciones **check** sobre una relación *r*, si la restricción tiene una subconsulta referenciando a *departamento*. Es razonable por la misma razón ya indicada para las restricciones de clave externa; una restricción **check** que referencia a una relación limita posibles actualizaciones sobre dicha relación.

4.6.5. Transferencia de privilegios

A un usuario al que se le haya concedido algún tipo de autorización puede permitírsele trasladar esa autorización a otros usuarios. De manera predeterminada, un privilegio que se concede a un usuario o a un rol no se puede trasladar a otros usuarios o roles. Si se desea conceder un privilegio y permitir a su receptor transferir dicho privilegio a otros usuarios, se añade la cláusula **with grant option** al comando **grant** correspondiente. Por ejemplo, si se desea conceder a Amit el privilegio **select** sobre *departamento* y permitir que Amit pueda conceder este privilegio a otros, se escribe:

grant select on *departamento* **to** Amit **with grant option**;

El creador de un objeto (relación/vista/rol) mantiene todos los privilegios sobre el objeto, incluyendo el privilegio de conceder privilegios a otros.

Considere, como ejemplo, la concesión de una autorización de actualización sobre la relación *enseña* de la base de datos de la universidad. Suponga que, inicialmente, el administrador de la base de datos concede autorización de actualización sobre *enseña* a los usuarios U_1, U_2 y U_3, quienes pueden conceder esta autorización a otros usuarios. El paso de una determinada autorización de un usuario a otro se puede representar mediante un **grafo de autorización.** Los nodos del grafo son los usuarios.

Considere el grafo de la autorización de actualización sobre *enseña*. El grafo incluye una arista $U_i \rightarrow U_j$ si el usuario U_i concede una autorización de actualización sobre *enseña* a U_j. La raíz del grafo es el administrador de la base de datos. En el grafo de ejemplo de la Figura 4.10, observe que al usuario U_5 se le concede una autorización tanto por parte de U_1 como de U_2; a U_4 se le concede una autorización por parte de U_1.

Un usuario tiene una autorización *si y solo si* existe un camino desde la raíz del grafo autorización (el nodo que representa al administrador de la base de datos) hasta el nodo que representa al usuario.

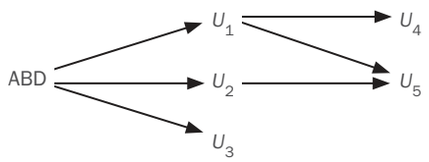


Figura 4.10. Grafo de concesión de autorización (U_1, U_2, …, U_5 son usuarios de la base de datos y ABD se refiere al administrador de la misma).

4.6.6. Revocación de privilegios

Suponga que el administrador de la base de datos decide revocar la autorización al usuario U_1. Como U_4 tiene una autorización de U_1, también se debería revocar esa autorización. Sin embargo, U_5 tenía concedida una autorización tanto de U_1 como de U_2. Como el administrador de la base de datos no revoca la autorización sobre *enseña* a U_2, U_5 mantiene su autorización sobre *enseña*. Si en algún momento U_2 revoca la autorización a U_5, entonces U_5 pierde la autorización.

Un par de usuarios maliciosos podrían intentar burlar las reglas de revocación de autorización concediéndose autorizaciones entre ellos. Por ejemplo, si U_2 recibe una concesión del administrador de la base de datos, y U_2 concede después autorización a U_3. Suponga que ahora U_3 retorna el privilegio de nuevo a U_2. Si el administrador de la base de datos revoca la autorización a U_2, podría parecer que U_2 mantiene la autorización a través de U_3. Sin embargo, tenga en cuenta que una vez que el administrador revoca la autorización a U_2 ya no existe ningún camino en el grafo de autorización desde la raíz ni a U_2 ni a U_3. Por tanto, SQL asegura que la autorización se revoca a ambos usuarios.

Como se ha visto, la revocación de un privilegio de un usuario o rol puede causar que otros usuarios o roles también lo pierdan. Este comportamiento se denomina *revocación en cascada*. En la mayoría de los sistemas de bases de datos esta cascada es el comportamiento predeterminado. Sin embargo, la sentencia **revoke** puede especificar la cláusula **restrict** para evitar que se produzca una revocación en cascada:

revoke select on *departamento* **from** Amit, Satoshi **restrict**;

En este caso, el sistema devuelve un error si se produce alguna revocación en cascada y no se lleva a cabo la acción de revocación.

Se puede utilizar la palabra clave **cascade** en lugar de **restrict** para indicar que la revocación se produce en cascada; sin embargo, se puede omitir, como se ha hecho en los ejemplos anteriores, ya que es el comportamiento predeterminado.

La siguiente sentencia **revoke** solo revoca la opción de concesión, en lugar del privilegio actual **select:**

revoke grant option for select on *departamento* **from** Amit;

Tenga en cuenta que algunas implementaciones de bases de datos no soportan la sintaxis anterior; en su lugar, se puede revocar el privilegio y volver a concederlo de nuevo sin la opción de concesión.

La revocación en cascada no es apropiada en muchas situaciones. Suponga que Satoshi tiene el rol *decano*, concede el rol *profesor* a Amit, y posteriormente se revoca el rol *decano* a Satoshi (quizás porque Satoshi abandona la universidad); Amit continúa como empleado de la facultad y, por tanto, debería mantener el rol de *profesor*.

Para tratar con esta situación, SQL permite que se conceda un privilegio a un rol en lugar de a un usuario. SQL tiene la noción de rol actual asociado con una sesión. De manera predeterminada, el rol actual asociado con una sesión es *null* (excepto en casos especiales). El rol actual asociado con una sesión se puede establecer ejecutando **ser role** *nombre_del_rol*. El rol especificado debe haberse concedido al usuario, o la sentencia **set role** fallará.

Para conceder un privilegio con un conjunto de concesiones al rol actual asociado a una sesión, se puede añadir la cláusula:

granted by current_role

a la sentencia de concesión, siempre que el rol actual no sea *null*.

Suponga que la concesión del rol *profesor* (u otros privilegios) a Amit se realiza usando la cláusula **granted by current_role,** con el rol actual de *decano*, en lugar de que a quien se le concede sea Satoshi. Entonces, la revocación de privilegios o roles (incluyendo el rol *decano*) a Satoshi no generará una revocación de privilegios que tuviese el conjunto de concesiones del rol *decano*, incluso aunque Satoshi fuese el usuario que ejecutó la concesión; por tanto, Amit mantendría el rol de *profesor* incluso aunque a Satoshi se le revocasen los privilegios.

4.7. Resumen

- SQL soporta varios tipos de reunión que incluyen la reunión interna y la reunión externa y diferentes tipos de condiciones de reunión.
- Las relaciones de vistas se definen como relaciones que contienen el resultado de las consultas. Las vistas resultan útiles para ocultar información innecesaria y para recopilar información de más de una relación en una única vista.
- Las transacciones son secuencias de consultas y de actualizaciones que, en su conjunto, desempeñan una tarea. Las transacciones se pueden comprometer o retroceder; cuando se hace retroceder una transacción, se deshacen los efectos de todas las actualizaciones llevadas a cabo por esa transacción.
- Las restricciones de integridad aseguran que los cambios realizados en la base de datos por usuarios autorizados no generan pérdidas ni inconsistencias de datos.
- Las restricciones de integridad referencial aseguran que un valor que aparece en una relación para un conjunto de atributos dado aparezca también para un conjunto de atributos concreto en otra relación.
- Las restricciones de dominio especifican el conjunto de valores posibles que pueden asociarse con cada atributo. Estas restricciones también pueden prohibir el uso de valores nulos para atributos concretos.
- Los asertos son expresiones declarativas que establecen predicados que se requiere que siempre sean ciertos.
- El lenguaje de definición de datos de SQL ofrece soporte para la definición de tipos predefinidos como **date** y **time,** así como para tipos de dominios definidos por los usuarios.
- Los mecanismos de autorización de SQL permiten diferenciar entre los usuarios de la base de datos, así como el tipo de acceso permitido para los distintos valores de datos de la base de datos.
- A un usuario al que se le ha concedido cierta forma de autoridad se le puede permitir transferir dicha autoridad a otros usuarios. Sin embargo, se debe tener cuidado sobre cómo se transfiere si se desea asegurar que estas autorizaciones se puedan revocar en el futuro.
- Los roles ayudan a asignar un conjunto de privilegios a los usuarios de acuerdo con el rol que el usuario desempeñe en la organización.

Términos de repaso

- Tipos de reunión.
  - Reunión interna y externa.
  - Reuniones externas por la izquierda, por la derecha y completa.
  - Natural, using y on.
- Definición de vistas.
- Vistas materializadas.
- Actualización de vistas.
- Transacciones.
  - Compromiso.
  - Retroceso.
  - Transacción atómica.
- Restricciones de integridad.
- Restricciones de dominios.
- Restricción de unicidad.
- Cláusula **check.**
- Integridad referencial.
  - Borrados en cascada.
  - Actualizaciones en cascada.
- Asertos.
- Tipos **date** y **time.**
- Valores predeterminados.
- Índices.
- Objetos grandes.
- Tipos definidos por el usuario.
- Dominios.
- Catálogos.
- Esquemas.
- Autorizaciones.
- Privilegios.
  - **select.**
  - **insert.**
  - **update.**
  - **all privileges.**
  - Concesión de privilegios.
  - Revocación de privilegios.
  - Privilegio de conceder privilegios.
  - Opción de concesión.
- Roles.
- Autorización sobre vistas.
- Ejecutar autorización.
- Invocador de privilegios.
- Autorización de fila.

Ejercicios prácticos

4.1. Escriba las siguientes consultas en SQL:

a. Mostrar una lista de todos los profesores, con su ID, nombre y número de sección en la que enseñan. Asegúrese de mostrar el número de sección como 0 para los profesores que no han enseñado en ninguna sección. La consulta debería utilizar una reunión externa y no utilizar subconsultas escalares.

b. Escriba la misma consulta anterior, pero utilizando una subconsulta escalar, sin reunión externa.

c. Mostrar la lista de todas las secciones de asignaturas ofertadas en la primavera de 2010, junto con los nombres de los profesores que enseñan en dichas secciones. Si una sección tiene más de un profesor, debería aparecer en el resultado tantas veces como profesores haya. Si no tiene ningún profesor, debería seguir apareciendo en el resultado con el nombre de profesor como «—».

d. Mostrar la lista de todos los departamentos, con el número total de profesores de cada uno, sin utilizar subconsultas escalares. Asegúrese de manejar correctamente los departamentos que no tienen profesores.

4.2. Las expresiones de reunión externa se pueden calcular en SQL sin utilizar la operación **outer join.** Para ello, demuestre cómo reescribir cada una de las siguientes consultas de SQL sin utilizar la expresión **outer join.**

a. **select*** **from** *estudiante* **natural left outer join** *matrícula*.

b. **select*** **from** *estudiante* **natural full outer join** *matrícula*.

4.3. Suponga que se tienen tres relaciones $r(A, B)$, $s(B, C)$ y $t(B, D)$, con todos los atributos declarados como **not null.** Considere las expresiones:

- r **natural left outer join** (s **natural left outer join** t), y
- (r **natural left outer join** s) **natural left outer join** t

a. Indique ejemplares de las relaciones r, s y t tales que en el resultado de la segunda expresión, el atributo C tenga un valor nulo pero el atributo D tenga un valor no nulo.

b. En el patrón anterior, ¿es posible que C sea nulo y D sea no nulo en el resultado de la primera expresión?

4.4. Prueba de consultas SQL: para comprobar si una consulta especificada en español se ha escrito correctamente en SQL, la consulta se ejecuta normalmente con varios ejemplos de bases de datos, y una persona comprueba si el resultado de la consulta de cada una de las pruebas coincide con lo pretendido en la especificación en español.

a. En la Sección 3.3.3 se vio un ejemplo de una consulta SQL errónea que pretendía encontrar qué cursos había enseñado cada profesor; la consulta calculaba la reunión natural de *profesor*, *enseña* y *asignatura*, y como resultado de forma no intencionada igualaba el atributo *nombre_dept* de *profesor* y *asignatura*. Indique un ejemplo de una base de datos que ayudaría a descubrir este error en concreto.

b. Cuando se crean bases de datos de prueba, es importante crear tuplas en relaciones referenciadas que no contienen ninguna tupla que case en la relación referenciada, para las claves externas. Explique por qué, utilizando un ejemplo de consulta en la base de datos de la universidad.

c. Cuando se crean bases de datos de prueba, es importante crear tuplas con valores nulos para atributos de clave externa, siempre que el atributo sea anulable (SQL permite que los atributos de clave externa incorporen valores nulos, con tal de que no formen parte de la clave primaria y no hayan sido declarados **not null**). Explique por qué, utilizando un ejemplo de consulta en la base de datos de la universidad.

(*Sugerencia*: utilice las consultas del Ejercicio 4.1).

4.5. Indique cómo definir la vista *estudiante_notas*(*ID*, *Prom*) indicando el promedio de cada uno de los estudiantes, de acuerdo con la consulta del Ejercicio 3.2; recuerde que se utiliza una relación *nota_puntos*(*nota*, *puntos*) para obtener los puntos numéricos asociados con la letra de la nota. Asegúrese de que la definición de la vista maneja correctamente el caso de los valores nulos en el atributo *nota* de la relación *enseña*.

4.6. Complete la definición del LDD de SQL de la base de datos de la Figura 4.8 para incluir las relaciones *estudiante*, *enseña*, *tutor* y *prerreq*.

4.7. Considere la base de datos relacional de la Figura 4.11. Indique una definición de esta base de datos en el LDD de SQL. Identifique las restricciones de integridad referencial que se deben cumplir e inclúyalas en la definición del LDD.

4.8. Como se ha tratado en la Sección 4.4.7, se espera que se cumpla la restricción «un profesor no puede enseñar secciones en dos aulas diferentes en un semestre en la misma franja horaria».

a. Escriba una consulta de SQL que devuelva todas las combinaciones (*profesor*, *sección*) que violan esta restricción.

b. Escriba un aserto de SQL que fuerce esta restricción (como se ha tratado en la Sección 4.4.7, los sistemas actuales de bases de datos no soportan tales asertos, aunque forman parte de la norma de SQL).

4.9. SQL permite que las dependencias de clave externa hagan referencia a la misma relación, como en el ejemplo siguiente:

```
create table jefe
(nombre_empleado   varchar(20) not null
nombre_jefe        varchar(20) not null,
primary key nombre_empleado,
foreign key (nombre_jefe) references jefe
on delete cascade)
```

Aquí, *nombre_empleado* es clave de la tabla *jefe*, lo que significa que cada empleado tiene como máximo un director. La cláusula de clave externa exige que cada director sea también empleado. Explique exactamente lo que ocurre cuando se borra una tupla de la relación *jefe*.

4.10. SQL-92 proporciona una operación n-aria denominada **coalesce** (fusión) que se define del modo siguiente: **coalesce**(A_1, A_2, …, A_n) devuelve el primer A_i no nulo de la lista A_1, A_2, …, A_n y devuelve un valor nulo si todos los valores son nulos.

Sean a y b relaciones con los esquemas A(*nombre*, *dirección*, *puesto*) y B(*nombre*, *dirección*, *sueldo*), respectivamente. Indique la manera de expresar a **natural full outer join** b, utilizando la operación **full outer-join** con una condición **on** y la operación **coalesce.** Compruebe que la relación resultado no contiene dos copias de los atributos *nombre* y *dirección* y que la solución es válida aunque alguna tupla de a o de b tenga valores nulos para los atributos *nombre* o *dirección*.

4.11. Algunos investigadores proponen el concepto de nulos *marcados*. Un nulo marcado $\perp_i$ es igual a el mismo, pero si $i \neq j$, entonces $\perp_i \neq \perp_j$. Una aplicación de los nulos marcados consiste en permitir ciertas actualizaciones mediante vistas. Considere la vista *profesor_info* (Sección 4.2).

Demuestre cómo se pueden utilizar los nulos marcados para permitir la inserción de la tupla (99999, «Johnson», «Música») mediante *profesor_info*.

empleado (*nombre_empleado*, *calle*, *ciudad*)
trabaja (*nombre_empleado*, *nombre_empresa*, *sueldo*)
empresa (*nombre_empresa*, *ciudad*)
jefe (*nombre_empleado*, *nombre_jefe*)

Figura 4.11. Base de datos empleada para los Ejercicios 4.7 y 4.12.

trabajador_jornada_completa (*nombre*, *despacho*, *teléfono*, *sueldo*)
trabajador_tiempo_parcial (*nombre*, *sueldo_por_hora*)
dirección (*nombre*, *calle*, *ciudad*)

Figura 4.12. Base de datos empleada para el Ejercicio 4.16.

Ejercicios

4.12. Para la base de datos de la Figura 4.11, escriba una consulta para encontrar aquellos empleados sin jefe. Tenga en cuenta que un empleado puede que no tenga un jefe asignado o puede que jefe valga *null*. Escriba una consulta utilizando una reunión externa y, después, escriba otra sin utilizar una reunión externa.

4.13. ¿Bajo qué circunstancias la siguiente consulta incluiría tuplas con valores nulos en el atributo *nombre_asig*?

```
select *
from estudiante natural full outer join matricula
     natural full outer join asignatura
```

4.14. Indique cómo definir una vista *total_créditos* (*año*, *núm_créditos*), que indica el número total de créditos en que se matricula un estudiante cada año.

4.15. Indique cómo expresar la operación **coales**ce del Ejercicio 4.10 utilizando la operación **case.**

4.16. Las restricciones de integridad referencial tal y como se han definido en este capítulo implican exactamente a dos relaciones. Considere una base de datos que incluye las relaciones que se muestran en la Figura 4.12. Suponga que se requiere que cada nombre que aparece en *dirección* aparezca también en *trabajador_jornada_completa* o en *trabajador_tiempo_parcial*, pero no necesariamente en ambos.

a. Proponga una sintaxis para expresar estas restricciones.

b. Explique las acciones que debe realizar el sistema para aplicar una restricción de este tipo.

4.17. Explique por qué, cuando un jefe, por ejemplo Satoshi, concede una autorización, la concesión se debería realizar utilizando el rol de jefe, y no como usuario Satoshi.

4.18. Suponga que un usuario A que tiene todas las autorizaciones sobre una relación r concede **select** sobre la relación r a **public** con opción de concesión. Suponga que entonces el usuario B concede **select** sobre r a A. ¿Genera esto un ciclo en el grafo de autorización? Explique por qué.

4.19. Los sistemas de bases de datos que almacenan cada relación en un archivo distinto del sistema operativo utilizan el esquema de autorización del sistema operativo, en lugar de definir un esquema especial propio. Discuta las ventajas y desventajas de este enfoque.

Notas bibliográficas

Consulte las notas bibliográficas del Capítulo 3 para obtener más información sobre SQL.

Las reglas que utiliza SQL para determinar la capacidad de actualización de una vista y cómo las actualizaciones se reflejan en las relaciones de la base de datos subyacente se definen en la norma SQL:1999 y se resumen en Melton y Simon [2001].

SQL avanzado

05

En los Capítulos 3 y 4 se trató con detalle la estructura básica de SQL. En este capítulo se van a tratar algunas de las características más avanzadas.[1]

Se verá cómo acceder a SQL desde lenguajes de programación de propósito general, lo que resulta muy importante para la creación de aplicaciones que usan una base de datos para almacenar y recuperar datos.

Se describe cómo ejecutar código procedimental en la base de datos, bien extendiendo el lenguaje SQL para soportar acciones procedimentales, o bien permitiendo definirlas en lenguajes procedimentales que se ejecuten en la base de datos.

Se describen los disparadores, que se pueden utilizar para especificar acciones a realizar de forma automática cuando se producen determinados eventos como la inserción, el borrado o la actualización de tuplas en una determinada relación.

Se verán las consultas recursivas y las características de agregación avanzada de SQL.

Finalmente, se describen los sistemas de procesamiento analítico en línea (OLAP), que permiten el análisis interactivo de grandes conjuntos de datos.

5.1. Acceso a SQL desde lenguajes de programación

SQL proporciona un lenguaje de consultas declarativo muy potente. La formulación de consultas en SQL es normalmente mucho más sencilla que la formulación de las mismas en un lenguaje de programación de propósito general. Sin embargo, los programadores deben tener acceso a la base de datos desde los lenguajes de programación de propósito general, al menos, por dos razones:

1. No todas las consultas pueden expresarse en SQL, ya que SQL no ofrece toda la potencia expresiva de los lenguajes de propósito general. Es decir, hay consultas que se pueden expresar en lenguajes como C, Java o Cobol que no se pueden expresar en SQL. Para formular consultas de este tipo, se puede incorporar SQL en un lenguaje más potente.
2. Las acciones no declarativas —como la impresión de informes, la interacción con los usuarios o el envío de los resultados de las consultas a una interfaz gráfica— no se pueden llevar a cabo desde el propio SQL. Normalmente, las aplicaciones contienen varios componentes y la consulta o actualización de los datos solo es uno de ellos; los demás componentes se escriben en lenguajes de programación de propósito general. En el caso de las aplicaciones integradas, los programas escritos en el lenguaje de programación deben poder tener acceso a la base de datos.

Existes dos enfoques sobre el acceso a SQL desde lenguajes de programación de propósito general:

- **SQL dinámico:** un programa de propósito general se puede conectar y comunicar con un servidor de base de datos utilizando una colección de funciones (en lenguajes procedimentales) o métodos (en lenguajes orientados a objetos). SQL dinámico permite al programa construir una consulta de SQL como una cadena de caracteres en tiempo de ejecución, enviar la consulta y obtener el resultado en variables del programa tupla a tupla. El componente *SQL dinámico* permite que los programas construyan y envíen consultas en tiempo de ejecución.

  En este capítulo se tratan dos normas para la conexión a una base de datos de SQL y la realización de consultas y actualizaciones. Una, JDBC (Sección 5.1.1), es una interfaz de programación de aplicaciones para el lenguaje Java. La otra, ODBC (Sección 5.1.2), es una interfaz de programación de aplicaciones, desarrollada originalmente para el lenguaje C y extendida después a otros lenguajes, como C++, C# y Visual Basic.
- **SQL embebido:** como el SQL dinámico, el SQL embebido proporciona un mecanismo para que los programas interactúen con una base de datos. Sin embargo, con SQL embebido las sentencias de SQL se identifican durante la compilación usando un preprocesador. El preprocesador envía las sentencias de SQL al sistema de base de datos para su precompilación y optimización; entonces se sustituyen las sentencias de SQL en el programa de aplicación con el código apropiado y las llamadas a funciones antes de iniciar la compilación del lenguaje de programación. En la Sección 5.1.3 se trata el SQL embebido.

El mayor desafío para mezclar SQL con un lenguaje de propósito general es la distinta forma en que ambos lenguajes manejan los datos. En SQL, el tipo de dato principal es la relación. Las sentencias de SQL operan sobre relaciones y devuelven relaciones como resultado. Los lenguajes de programación normalmente operan con una variable cada vez, y dichas variables se corresponden de modo grueso con el valor de un atributo en una tupla de una relación. Por tanto, la integración de estos dos tipos de lenguajes en una única aplicación requiere proporcionar mecanismos que devuelvan el resultado de una consulta de forma que se puedan manejar desde el programa.

5.1.1. JDBC

La norma **JDBC** define una **interfaz de programación de aplicaciones** (*application program interface*: API) que pueden utilizar los programas en Java para conectarse a servidores de bases de datos. La palabra JDBC era originalmente una abreviatura de **Java Database Connectivity** (Java para conectividad de base de datos), pero ya no se utiliza en su forma completa.

En la Figura 5.1 se muestra un ejemplo de programa en Java que usa la interfaz JDBC. Muestra cómo se abre una conexión, se ejecuta y se procesan los resultados, y se cierra la conexión. Se verá con detalle este ejemplo en esta sección. El programa en Java debe importar `java.sql.*`, que contiene la definición de interfaces para la funcionalidad de JDBC.

1 **Nota sobre la secuencia de capítulos y secciones:** el diseño de bases de datos, Capítulos 7 y 8, se puede estudiar de manera independiente al material de este capítulo. Es posible estudiar primero el diseño de bases de datos y este capítulo después. Sin embargo, para los cursos que ponen énfasis en la programación, se pueden realizar muchos más ejercicios de laboratorio tras estudiar la Sección 5.1, y se recomienda empezar el estudio por ella antes del diseño de bases de datos.

5.1.1.1. Conexión a la base de datos

El primer paso para acceder a una base de datos desde un programa en Java es abrir una conexión a la misma. Este paso es un requisito para seleccionar la base de datos que se va a utilizar, por ejemplo, un ejemplar de Oracle que se ejecuta en su máquina, o una base de datos PostgreSQL que se ejecuta en otra máquina. Solo después de abrir una conexión el programa Java puede ejecutar sentencias de SQL.

Para abrir una conexión se usa el método getConnection de la clase DriverManager (de java.sql). Este método tiene tres parámetros.[2]

- El primer parámetro de la llamada a getConnection es un cadena de caracteres (*string*) que especifica la URL, o nombre de la máquina en la que se ejecuta el servidor (en el ejemplo, db.yale.edu), junto con, posiblemente, otra información, como el protocolo a utilizar para comunicarse con la base de datos (en el ejemplo, jdbc:oracle:thin:, en breve se verá por qué es necesario), el número de puerto que utiliza el sistema de base de datos para comunicarse (en el ejemplo, 2000) y la base de datos concreta en el servidor que se va a utilizar (en el ejemplo, univdb). Tenga en cuenta que JDBC solo especifica la API, no el protocolo de comunicaciones. Un controlador de JDBC puede disponer de varios protocolos, y hay que especificar uno de los que admite tanto la base de datos como el controlador. Los detalles del protocolo dependen del fabricante.
- El segundo parámetro de getConnection es un identificador de usuario de la base de datos, que también es una cadena de caracteres (*string*).
- El tercer parámetro es una contraseña, que también es una cadena de caracteres (*string*). (Tenga en cuenta que el tener que indicar una contraseña dentro del código implica un riesgo de seguridad si una persona no autorizada accede al código en Java).

En el ejemplo de la figura se ha creado un objeto Connection cuyo manejador se llama conn.

Los productos de bases de datos de que dispone JDBC (todos los principales fabricantes de bases de datos) proporcionan un controlador de JDBC que se debe cargar de forma dinámica para poder acceder a la base de datos desde Java. De hecho, el controlador debe cargarse al principio, antes de conectarse a la base de datos.

Esto se hace invocando Class.forName con un argumento que especifica la clase concreta que implementa la interfaz java.sql.Driver, en la primera línea del programa de la Figura 5.1. Esta interfaz proporciona la traducción de llamadas de JDBC independientes del producto en llamadas dependientes del producto requeridas por el sistema de gestión de bases de datos concreto que se esté utilizando. El ejemplo de la figura muestra el controlador de Oracle, oracle.jdbc.driver.OracleDriver.[3] El controlador está disponible en un archivo .jar en el sitio web del fabricante y debería ubicarse en el classpath para que el compilador de Java pueda encontrarlo.

El protocolo real que se usa para cambiar información con la base de datos depende del controlador utilizado, y no viene definido por la norma JDBC. Algunos controladores soportan más de un protocolo y se elige el apropiado dependiendo de qué protocolo admita la base de datos a la que se conectan. En el ejemplo, cuando se abre una conexión con la base de datos, la cadena de caracteres jdbc:oracle:thin: especifica un protocolo concreto que admite Oracle.

2 Existen varias formas del método getConnection, que se diferencian por los parámetros que aceptan. Se presenta el más utilizado habitualmente.

3 Los nombres de los controladores equivalentes para otros productos son los siguientes: para DB2 de IBM: com.ibm.db2.jdbc.app.DB2Driver; para SQL Server de Microsoft: com.microsoft.sqlserver.jdbc.SQLServerDriver; para PostgreSQL: org.postgresql.Driver, y para MySQL: com.mysql.jdbc.Driver. Sun también ofrece un «controlador puente» que convierte las llamadas de JDBC en ODBC. Este último solo se debe usar con fabricantes que soporten ODBC pero no JDBC.

5.1.1.2. Envío de sentencias de SQL al sistema de bases de datos

Una vez abierta una conexión a la base de datos, el programa puede utilizarla para enviar sentencias de SQL al sistema de bases de datos para su ejecución. Esto se hace mediante un ejemplar de la clase Statement. Un objeto Statement no es una sentencia de SQL en sí misma, sino un objeto que permite que los programas de Java invoquen los métodos que envían una sentencia de SQL como argumento para su ejecución en el sistema de bases de datos. El ejemplo crea un manejador de Statement (stmt) para la conexión conn.

Para ejecutar una sentencia, se invoca el método executeQuery o el método executeUpdate, dependiendo de si la sentencia de SQL es una consulta (y por tanto devuelve un resultado) o no es una consulta, como **update, insert, delete, create table,** etc. En el ejemplo, stmt.executeUpdate ejecuta una sentencia de actualización que realiza una inserción en la relación *profesor*. Devuelve un entero que indica el número de tuplas insertadas, actualizadas o borradas. Para las sentencias de LDD, el valor devuelto es cero. La construcción try {...} catch {...} permite capturar las excepciones (condiciones de error) que se puedan producir cuando se realiza la llamada de JDBC y se imprime un mensaje apropiado al usuario.

5.1.1.3. Obtención del resultado de una consulta

El programa de ejemplo ejecuta una consulta utilizando stmt.executeQuery. Recupera el conjunto de tuplas del resultado en el objeto rset de la clase ResultSet tupla a tupla. El método next sobre el conjunto de resultados comprueba si existen más tuplas pendientes en el conjunto de resultados, y si es así, las obtiene. El valor devuelto por el método next es un booleano que indica si se ha obtenido una tupla. Los atributos de la tupla obtenida se consiguen utilizando varios métodos cuyo nombre empieza por get. El método getString puede obtener cualquiera de los tipos de datos básicos de SQL (convertidos los valores en un objeto String de Java), pero también se pueden utilizar métodos restrictivos como getFloat. El argumento de distintos métodos get puede ser un nombre de atributo especificado como una cadena de texto (*string*) o un entero que indica la posición del atributo deseado en la tupla. La Figura 5.1 muestra dos formas de recuperar los valores de los atributos de las tuplas: empleando el nombre del atributo (*nombre_dept*) y utilizando la posición del atributo (2, para indicar el segundo atributo).

La sentencia y la conexión se cierran al final del programa de Java. Observe que es importante cerrar la conexión, ya que se ha impuesto un límite al número de conexiones con la base de datos; las conexiones no cerradas pueden hacer que ese límite se supere. Si esto ocurre, la aplicación no podrá abrir más conexiones con la base de datos.

5.1.1.4. Sentencias preparadas

Se pueden crear sentencias preparadas en las que algunos valores sean sustituidos por «?», lo que indica que los valores reales se proporcionarán más tarde. Los sistemas de bases de datos compilan las consultas cuando se preparan. Cada vez que se ejecuta la consulta (con valores nuevos que sustituyen las «?»), la base de datos puede volver a emplear la forma previamente compilada de la consulta y aplicar los nuevos valores. El fragmento de código de la Figura 5.2 muestra la forma de utilizar las sentencias preparadas.

El método prepareStatement de la clase Connection envía una sentencia de SQL para su compilación. Devuelve un objeto de la clase PreparedStatement. En este momento todavía no se ha ejecutado ninguna sentencia de SQL. Los métodos executeQuery y executeUpdate de la clase PreparedStatement son los que lo hacen.

Pero antes de poder llamarlos se deben utilizar métodos de la clase PreparedStatement para asignar los valores a los parámetros «?». El método setString y otros métodos similares, como setInt para otros tipos básicos de SQL, permiten especificar los valores de los parámetros. El primer argumento especifica el parámetro «?» para el que se asigna un valor (el primer parámetro es el 1, al contrario que en otras construcciones de Java que empiezan por el 0). El segundo argumento especifica el valor a asignar.

En el ejemplo de la figura, se prepara una sentencia **insert,** se establecen los parámetros «?» y después se invoca a executeUpdate. Las dos líneas finales del ejemplo muestran que los parámetros se mantienen hasta que no se modifiquen explícitamente. Por tanto, la sentencia final, que invoca a executeUpdate, inserta la tupla («88878», «Perry», «Finanzas», 125000).

Las sentencias preparadas permiten una ejecución más eficiente en aquellos casos en que la misma consulta se puede compilar una vez y ejecutar después muchas veces con diferentes valores de los parámetros. Sin embargo, existe una ventaja más significativa de las sentencias preparadas que las convierte en el método preferido para la ejecución de consultas de SQL siempre que se utilicen valores introducidos por los usuarios, incluso aunque la consulta se ejecute una sola vez. Suponga que se lee un valor que introduce un usuario y después se manipula una cadena de caracteres de Java para construir una sentencia de SQL. Si el usuario introduce ciertos caracteres especiales, como una comilla simple, el resultado de la sentencia puede ser sintácticamente incorrecto a no ser que se tenga un extraordinario cuidado en la comprobación de la entrada. El método setString lo hace automáticamente e inserta los caracteres de escape necesarios para asegurar la corrección sintáctica.

En el ejemplo, suponga que el valor de las variables *ID*, *nombre*, *nombre_dept* y *sueldo* los ha introducido el usuario y hay que insertar la fila correspondiente en la relación *profesor*. Suponga que, en lugar de utilizar una sentencia preparada, se crea una consulta mediante la concatenación de las cadenas de caracteres utilizando la expresión de Java:

“insert into profesor values(´ “ + ID +” ´,´”+ nombre + “ ´,”+
“´+ nombre_dept + “ ´, “ ´ sueldo + “)”

y la consulta se ejecuta directamente usando el método executeQuery de un objeto Statement. Ahora, si el usuario escribiera una comilla simple en el campo *ID* o en el del nombre, la cadena de caracteres de la consulta tendría un error de sintaxis. Es posible que algún nombre de profesor tenga un apóstrofo (especialmente si es un nombre irlandés, como O´Donnell).

Aunque la situación anterior puede considerarse problemática, puede ser aún peor. Se puede utilizar una técnica denominada **inyección SQL** propia de hackers maliciosos para robar datos o crear daños en la base de datos.

Suponga que un programa Java recibe un string *nombre* y construye la consulta:

“select * from profesor where nombre = ´” + nombre + “´”

Si el usuario en lugar de escribir un nombre escribe:

X´ or ´Y´ = ´Y

entonces la sentencia resultado se convierte en:

“select * from profesor where nombre = ´” + “X´ or ´Y´ = ´Y” + “´”

que es:

select * from profesor where nombre = ´X´ or ´Y´ = ´Y´

En la consulta resultado, la cláusula **where** es siempre cierta y se devuelve la relación profesor completa. Usuarios maliciosos más inteligentes pueden conseguir incluso más datos. El uso de una sentencia preparada podría evitar este problema, ya que la cadena de entrada tendría caracteres de escape insertados, por lo que la consulta se convertiría en:

“select * from profesor where nombre = ´X\´ or \´Y\´= \´Y´

que no es peligrosa y devuelve una relación vacía.

Los sistemas antiguos permiten ejecutar varias sentencias en una única llamada, con sentencias separadas por punto y coma. Esta característica se está eliminando debido a la técnica de inyección SQL utilizada por hackers maliciosos para insertar sentencias de SQL completas. Como estas sentencias se ejecutan en Java con los privilegios del propietario del programa, se podrían ejecutar sentencias SQL devastadoras, como **drop table.** Los desarrolladores de aplicaciones en SQL necesitan ser conscientes de estos potenciales agujeros de seguridad.

```
public static void JDBCejemplo(String usuarioid, String contraseña)
{
  try
  {
    Class.forName (“oracle.jdbc.driver.OracleDriver”);
    Connection conn = DriverManager.getConnection(
      “jdbc:oracle:thin:@db.yale.edu:1521:univdb”,
       usuarioid, contraseña);
    Statement stmt = conn.createStatement();
    try {
      stmt.executeUpdate(
        “insert into profesor values(´77987´, ´Kim´,
         ´Física´, 98000)”);
    } catch (SQLException sqle)
    {
      System.out.println(“No se pudo insertar la tupla. “ + sqle);
    }
    ResultSet rset = stmt.executeQuery(
      “select nombre_dept, avg (sueldo) “+
      “ from profesor “+
      “ group by nombre_dept “);
    while (rset.next()) {
      System.out.println(rset.getString(“nombre_dept “) +
      ” ”+ rset.getFloat(2));
    }
    stmt.close();
    conn.close();
  }
  catch (Exception sqle)
  {
    System.out.println(“Excepción : “ + sqle);
  }
}
```

Figura 5.1. Ejemplo de código de JDBC.

```
PreparedStatement pStmt = conn.prepareStatement(
                    “insert into profesor values(?,?,?,?)”);
pStmt.setString(1, “88877”);
pStmt.setString(2, “Perry”);
pStmt.setString(3, “Finanzas”);
pStmt.setInt(4, 125000);
pStmt.executeUpdate();
pStmt.setString(1, “88878”);
pStmt.executeUpdate();
```

Figura 5.2. Sentencias preparadas en el código JDBC.

5.1.1.5. Sentencias que se pueden invocar

JDBC también proporciona una interfaz CallableStatement que permite la invocación de los procedimientos y funciones almacenados de SQL (que se describen más adelante, en la Sección 5.2). Esta interfaz desempeña el mismo rol para las funciones y los procedimientos que prepareStatement para las consultas.

```
CallableStatement cStmt1 =
        conn.prepareCall("{? = call una_función(?)}");
CallableStatement cStmt2 =
        conn.prepareCall("{call un_procedimiento(?,?)}");
```

Los tipos de datos de los valores devueltos por las funciones y de los parámetros externos de los procedimientos deben registrarse empleando el método registerOutParameter(), y pueden recuperarse utilizando métodos get parecidos a los de los conjuntos de resultados. Consulte el manual de JDBC para obtener más información.

5.1.1.6. Metadatos

Como se ha indicado anteriormente, un programa de aplicación Java no incluye declaraciones para datos almacenados en la base de datos. Estas declaraciones forman parte de las sentencias de LDD de SQL. Por tanto, un programa Java que usa JDBC debe realizar suposiciones sobre el esquema de la base de datos codificador en el propio programa o determinar la información directamente de la base de datos en tiempo de ejecución. El último enfoque suele ser preferible, ya que el programa de aplicación será más robusto frente a posibles cambios en el esquema de la base de datos.

Recuerde que cuando se envía una consulta usando el método executeQuery, el resultado de la consulta se obtiene en un objeto ResultSet. La interfaz ResultSet tiene un método getMetaData() para obtener objetos ResultSetMetaData y proporcionar los metadatos del conjunto de resultados. ResultSetMetaData, a su vez, contiene métodos para determinar la información de los metadatos, como puede ser el número de columnas de un resultado, el nombre de una columna concreta o el tipo de datos de una columna dada. De esta forma se puede ejecutar una consulta incluso aunque no se conozca el esquema del resultado.

El fragmento de programa de Java que se muestra a continuación utiliza JDBC para escribir el nombre y el tipo de datos de todas las columnas de un conjunto de resultados. La variable rs en el código se supone que se refiere a un ejemplar de ResultSet que se obtiene de la ejecución de una consulta.

```
ResultSetMetaData rsmd = rs.getMetaData();
for(int i = 1; i <= rsmd.getColumnCount(); i++) {
    System.out.println(rsmd.getColumnName(i));
    System.out.println(rsmd.getColumnTypeName(i));
}
```

El método getColumnCount devuelve la cardinalidad (número de atributos) de la relación resultado. Esto nos permite iterar por los atributos (observe que se empieza por 1, como es convenio en JDBC). Para cada atributo se obtiene su nombre y el tipo de dato utilizando los métodos getColumnName y getColumnTypeName, respectivamente.

La interfaz DatabaseMetaData ofrece una forma de determinar los metadatos de la base de datos. La interfaz Connection tiene un método getMetaData que devuelve el objeto DatabaseMetaData. La interfaz DatabaseMetaData, a su vez, contiene gran número de métodos para obtener los metadatos de la base de datos a la que está conectada la aplicación.

Por ejemplo, existen métodos que devuelven el nombre del producto y el número de versión del sistema de base de datos. Otros métodos permiten a la aplicación obtener información sobre las características soportadas.

Además, existen métodos para obtener información sobre la propia base de datos. El código de la Figura 5.3 muestra la manera de obtener información sobre las columnas (atributos) de las relaciones de las bases de datos. La variable conn se supone que es un manejador para una conexión de base de datos ya abierta. El método getColumns tiene cuatro argumentos: un nombre de catálogo (*null* significa que el nombre de catálogo se ignore), un nombre de patrón de esquema, un nombre de patrón de tabla y un nombre de patrón de columna. Los nombre de los patrones de esquema, tabla y columna se utilizan para especificar un nombre o un patrón. Los patrones pueden usar cadenas de SQL que casan los caracteres especiales «%» y «_»; por ejemplo, el patrón «%» casa con todos los nombres. Solo se devuelven las columnas de las tablas de los esquemas que satisfagan los nombres o los patrones indicados. Las filas tienen un número de columnas, así como el nombre del catálogo, el esquema, la tabla y la columna, el tipo de columna, etc.

Otros ejemplos de métodos de DatabaseMetaData que proporcionan información sobre la base de datos incluyen aquellos para obtener los metadatos sobre las relaciones (getTables()), referencias de claves externas (getCrossReference()) y otros.

Las interfaces de metadatos se pueden utilizar para distintas tareas. Por ejemplo, se pueden utilizar para escribir un explorador de la base de datos que permita al usuario descubrir las tablas de la base de datos, examinar su esquema, examinar las filas de una tabla, aplicar selecciones sobre las filas deseadas, etc. La información de los metadatos se puede utilizar para que el código de estas tareas sea genérico; por ejemplo, el código para mostrar las filas de una relación se puede escribir de forma que funcione con cualquier relación, independientemente de su esquema. De forma similar, es posible escribir código que recoja una cadena de consulta, ejecute la consulta y muestre el resultado en una tabla con formato; el código puede funcionar independientemente de la consulta enviada.

5.1.1.7. Otras características

JDBC ofrece varias características más, como los **conjuntos de resultados actualizables.** Se pueden crear conjuntos de resultados actualizables a partir de consultas que lleven a cabo selecciones y/o proyecciones de las relaciones de las bases de datos. La actualización de una tupla del conjunto de resultados tiene como consecuencia que se actualice la tupla correspondiente de la relación de la base de datos.

Recuerde de la Sección 4.3 que una transacción permite tratar varias acciones como una única unidad atómica que se puede comprometer o retroceder.

De manera predeterminada, cada instrucción de SQL se trata como una transacción independiente que se compromete de manera automática. El método setAutoCommit() de la interfaz Connection de JDBC permite que este comportamiento se active o se desactive. Por tanto, si conn es una variable que almacena una conexión abierta, conn.setAutoCommit(false) desactivará el compromiso automático. Las transacciones, entonces, deben comprometerse de manera explícita mediante con.commit() o retrocederse mediante con.rollback(). El compromiso automático puede activarse mediante con.setAutoCommit(true).

JDBC proporciona interfaces para el tratamiento de objetos de gran tamaño sin necesidad de crearlos en la memoria. Para obtener objetos de gran tamaño la interfaz ResultSet proporciona los métodos getBlob() y getClob(), que son parecidos al método getString() pero devuelven objetos de los tipos de datos Blob y Clob, respectivamente. Estos objetos no almacenan los objetos de gran tamaño completos, sino sus «localizadores», es decir, punteros lógicos a los objetos grandes reales en la base de datos. La obtención de los datos de estos objetos es muy similar a obtener los datos de un archivo o un flujo de entrada, y se puede realizar utilizando métodos como getBytes y getSubString.

De forma análoga, para guardar grandes objetos en la base de datos, la clase PreparedStatement permite que una columna de datos cuyo tipo es **blob** se enlace con un flujo de entrada (como un archivo que se haya abierto) utilizando el método setBlob(int parameterIndex, InputStream inputStream). Cuando la sentencia preparada se ejecuta, los datos se leen del flujo de entrada y se escriben en el **blob** en la base de datos. De la misma forma, una columna **clob** se puede establecer con el método setClob, que tiene como argumentos un parámetro índice y un flujo de caracteres.

JDBC también incluye la característica de *conjunto de filas*, que permite que los conjuntos de resultados se recojan y envíen a otras aplicaciones. Los conjuntos de filas se pueden analizar hacia delante y hacia atrás y se pueden modificar. Como los conjuntos de filas una vez descargados no forman parte de la propia base de datos, no se tratan con detalle en este libro.

5.1.2. ODBC

La norma ODBC (*open database connectivity*: **conectividad abierta de bases de datos**) define una API que pueden utilizar las aplicaciones para abrir conexiones con una base de datos, enviar consultas y actualizaciones y obtener resultados. Las aplicaciones como interfaces gráficas de usuario, paquetes estadísticos y hojas de cálculo pueden hacer uso de la misma API de ODBC para conectarse a cualquier servidor de bases de datos compatible con ODBC.

Cada sistema de bases de datos compatible con ODBC proporciona una biblioteca que se debe enlazar con el programa cliente. Cuando el programa cliente realiza una llamada a la API de ODBC, el código de la biblioteca se comunica con el servidor para llevar a cabo la acción solicitada y obtener los resultados.

La Figura 5.4 muestra un ejemplo de código en C que usa la API de ODBC. El primer paso en ODBC para comunicarse con un servidor consiste en configurar su conexión. Para ello, el programa asigna en primer lugar un entorno de SQL y posteriormente un manejador para la conexión a la base de datos. ODBC define los tipos HENV, HDBC y RETCODE. El programa abre a continuación la conexión a la base de datos empleando SQLConnect. Esta llamada tiene varios parámetros, como son el manejador de la conexión, el servidor al que hay que conectarse, el identificador de usuario y la contraseña para la base de datos. La constante SQL_NTS indica que el argumento anterior es una cadena terminada en un valor nulo.

Una vez configurada la conexión, el programa puede enviar comandos SQL a la base de datos empleando SQLExecDirect. Las variables del lenguaje C se pueden vincular a los atributos del resultado de la consulta, de forma que cuando se obtenga una tupla resultado mediante SQLFetch, los valores de sus atributos se almacenen en las variables de C correspondientes. La función SQLBindCol realiza esta tarea; el segundo argumento identifica la posición del atributo en el resultado de la consulta, y el tercer argumento indica la conversión requerida de tipos de SQL a C. El siguiente argumento proporciona la dirección de la variable. Para los tipos de datos de longitud variable, como los arrays de caracteres, los dos últimos argumentos proporcionan la longitud máxima de la variable y una ubicación en la que se debe almacenar la longitud real cuando se obtenga una tupla. Un valor negativo devuelto para el argumento de la longitud indica que el valor es **null.** Para los tipos de datos de longitud fija como integer o float, se ignora el argumento que indica la longitud máxima, mientras que la devolución de un valor negativo para el argumento de la longitud indica un valor nulo.

La sentencia SQLFetch está en un bucle **while** que se ejecuta hasta que SQLFetch devuelva un valor diferente de SQL_SUCCESS. En cada iteración, el programa almacena los valores en variables de C como se especifica mediante las llamadas a SQLBindCol e imprime esos valores.

Al final de la sesión, el programa libera el controlador de la instrucción, se desconecta de la base de datos y libera la conexión y los manejadores del entorno de SQL. Un buen estilo de programación exige que el resultado de cada llamada a una función se compruebe para asegurarse de que no haya errores; se han omitido la mayoría de estas comprobaciones por brevedad.

Es posible crear sentencias de SQL con parámetros; por ejemplo, considere la sentencia insert into departamento values(?,?,?). Los signos de interrogación representan los valores que se proporcionarán después. Esta sentencia se puede «preparar», es decir, compilar en la base de datos y ejecutar repetidamente proporcionando los valores reales para los parámetros, en este caso, proporcionando un nombre de departamento, un edificio y un presupuesto para la relación *departamento*.

```
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet rs = dbmd.getColumns(null, "univdb", "departamento", "%");
  // Argumentos de getColumns: Catálogo, Patrón-esquema,
     Patrón-tabla y Patrón-columna
  // Devuelve: Una fila por columna; la fila tiene un número
     de atributos como COLUMN_NAME, TYPE_NAME
while(rs.next()) {
  System.out.println(rs.getString("COLUMN_NAME"),
                     rs.getString("TYPE_NAME");
}
```

Figura 5.3. Obtener la información de columnas en JDBC utilizando DatabaseMetaData.

```
void ODBCejemplo()
{
  RETCODE error;
  HENV env; /* entorno */
  HDBC conn; /* conexión a la base de datos */

  SQLAllocEnv(&env);
  SQLAllocConnect(env, &conn);
  SQLConnect(conn, "db.yale.edu", SQL_NTS, "avi", SQL_NTS,
             "avicontraseña", SQL_NTS);
  {
    char nombredept [80];
    float sueldo;
    int lenOut1, lenOut2;
    HSTMT stmt;

    char * sqlquery = "select nombre_dept, sum (sueldo)
                       from profesor
                       group by nombre_dept ";
    SQLAllocStmt(conn, &stmt);
    error = SQLExecDirect(stmt, sqlquery, SQL_NTS);
    if (error == SQL_SUCCESS) {
      SQLBindCol(stmt, 1, SQL_C_CHAR, nombredept, 80, &lenOut1);
      SQLBindCol(stmt, 2, SQL_C_FLOAT, &sueldo, 0 , &lenOut2);
      while (SQLFetch(stmt) == SQL_SUCCESS) {
          printf (" %s %g\n", nombredept, sueldo);
      }
    }
    SQLFreeStmt(stmt, SQL_DROP);
  }
  SQLDisconnect(conn);
  SQLFreeConnect(conn);
  SQLFreeEnv(env);
}
```

Figura 5.4. Código de ejemplo de ODBC.

ODBC define funciones para gran variedad de tareas, tales como encontrar todas las relaciones de la base de datos y los nombres y tipos de las columnas del resultado de una consulta o de una relación de la base de datos.

De forma predeterminada, cada sentencia de SQL se trata como una transacción separada que se compromete automáticamente. La llamada SQLSetConnectOption(con, SQL_AUTOCOMMIT, 0) desactiva el compromiso automático en la conexión conn, por lo que las transacciones se deben comprometer explícitamente mediante SQLTransact(con, SQL_COMMIT) o retroceder mediante SQLTransact(con, SQL_ROLLBACK).

La norma ODBC define *niveles de conformidad*, que especifican subconjuntos de la funcionalidad definida por la norma. Puede que una implementación de ODBC solo proporcione las características básicas, o puede proporcionar características más avanzadas (de nivel 1 o de nivel 2). El nivel 1 exige soporte para la captura de información sobre el catálogo, como puede ser la información sobre las relaciones existentes y los tipos de sus atributos. El nivel 2 exige más características, como la capacidad de enviar y obtener arrays de valores de parámetros y la de obtener información del catálogo más detallada.

La norma de SQL define una **interfaz del nivel de llamadas** (*call level interface*: CLI) que es parecida a la interfaz de ODBC.

ADO.NET

La API de ADO.NET, diseñada para los lenguajes Visual Basic .NET y C#, proporciona funciones de acceso a datos, que en gran medida no son muy distintas de las funciones de JDBC, aunque difieren en los detalles. Como JDBC y ODBC, la API de ADO.NET permite acceder a los resultados de las consultas de SQL así como a los metadatos, pero es considerablemente más sencillo de usar que ODBC. Con ADO.NET se puede acceder a una base de datos que soporta ODBC, y las llamadas de ADO.NET se traducen a llamadas de ODBC. La API de ADO.NET también se puede usar con algunos tipos de fuentes de datos no relacionales como OLE-DB de Microsoft, XML (se trata en el Capítulo 22) y, más recientemente, con el Entity Framework desarrollado por Microsoft. Consulte la bibliografía para más información sobre ADO.NET.

5.1.3. SQL incorporado

La norma SQL define la incorporación de SQL en varios lenguajes de programación, tales como C, C++, Cobol, Pascal, Java, PL/I y Fortran. El lenguaje en el que se incorporan las consultas SQL se denomina lenguaje *anfitrión* y las estructuras de SQL que se admiten en el lenguaje anfitrión constituyen SQL *incorporado*.

Los programas escritos en el lenguaje anfitrión pueden usar la sintaxis de SQL incorporado para actualizar y tener acceso a los datos almacenados en la base de datos. Un programa con SQL incorporado debe ser procesado por un preprocesador antes de su compilación. El preprocesador sustituye el SQL incorporado con declaraciones y llamadas a procedimientos del lenguaje anfitrión que permitan la ejecución en tiempo de ejecución de los accesos a la base de datos. Entonces, el programa resultado se compila en el compilador del lenguaje anfitrión. Esta es la principal diferencia entre SQL incorporado y JDBC u ODBC.

En JDBC las sentencias de SQL se interpretan en tiempo de ejecución (incluso aunque estén preparadas antes, usando la característica de sentencias preparadas). Cuando se usa SQL incorporado, se pueden capturar algunos errores relacionados con SQL (incluyendo los errores de tipos de datos) en tiempo de compilación.

Para identificar las consultas al preprocesador de SQL incorporado se utiliza la instrucción EXEC SQL; que tiene la forma:

EXEC SQL <instrucción de SQL incorporado>;

La sintaxis exacta de las consultas de SQL incorporado depende del lenguaje en el que se haya incorporado SQL. En algunos lenguajes, como en Cobol, el punto y coma se sustituye por END-EXEC.

En el programa se incluye la sentencia SQL INCLUDE SQLCA para identificar el lugar en que el preprocesador debe insertar las variables especiales que se emplean para la comunicación entre el programa y el sistema de bases de datos.

Antes de ejecutar ninguna sentencia de SQL el programa debe conectarse con la base de datos. Esto se logra mediante:

EXEC SQL **connect to** *servidor*
user *nombre-usuario* **using** *contraseña*;

En este caso, *servidor* identifica al servidor con el que hay que establecer la conexión.

Se pueden utilizar las variables del lenguaje anfitrión con las sentencias de SQL incorporadas, pero deben estar precedidas por dos puntos (:) para distinguirlas de las variables de SQL. Las variables utilizadas de esta forma se deben declarar en una sección DECLARE, como se muestra a continuación. Sin embargo, la sintaxis para declarar variables sigue la sintaxis usual del lenguaje anfitrión:

EXEC SQL BEGIN DECLARE SECTION;
int *cantidad_créditos*;
EXEC SQL END DECLARE SECTION;

Las sentencias de SQL incorporado son parecidas en cuanto a su forma a las instrucciones de SQL normales. Sin embargo, hay varias diferencias que se indican a continuación.

Para escribir una consulta relacional se emplea la sentencia **declare cursor.** El resultado de la consulta no se calcula todavía. En su lugar, el programa debe utilizar los comandos **open** y **fetch** (que se analizarán más adelante en esta sección) para obtener las tuplas resultado. Como se verá, el uso de un cursor es análogo a la iteración por el conjunto de resultados en JDBC.

Considere el esquema de la universidad. Suponga que se tiene en el programa del lenguaje anfitrión la variable *cantidad_créditos*, declarada como se ha visto antes, y se quieren encontrar los nombres de todos los estudiantes que se han matriculado en más horas de crédito de las que marca la variable *cantidad_créditos*. Se puede escribir esta consulta como:

EXEC SQL
declare *c* **cursor for**
select *ID*, *nombre*
from *estudiante*
where *tot_créditos* > :*cantidad_créditos*;

La variable *c* de la expresión anterior se denomina *cursor* de la consulta. Esta variable se utiliza para identificar la consulta. Posteriormente se usa la sentencia **open,** que hace que se evalúe la consulta.

La sentencia **open** para la consulta anterior es:

EXEC SQL **open** *c*;

Esta sentencia hace que el sistema de base de datos ejecute la consulta y guarde el resultado en una relación temporal. La consulta utiliza el valor de la variable del lenguaje anfitrión (*cantidad_créditos*) en el momento en que se ejecuta la sentencia **open.**

Si la consulta de SQL genera un error, el sistema de base de datos almacena un diagnóstico de error en las variables del área de comunicación de SQL (SQLCA).

A continuación se ejecutan una serie de sentencias **fetch;** cada una obtiene un valor que se sitúa en una variable del lenguaje anfitrión. La instrucción **fetch** necesita una variable del lenguaje anfitrión por cada atributo de la relación resultado. En la consulta de ejemplo se necesita una variable para almacenar el valor del *ID* y otra para el valor del *nombre*. Suponga que esas variables son *vi* y *vn*, respectivamente. Entonces, la sentencia:

EXEC SQL **fetch** *c* **into** :*vi*, :*vn*;

genera una tupla de la relación resultado. El programa puede manipular entonces las variables *vi* y *vn* empleando las características del lenguaje de programación anfitrión.

Cada solicitud **fetch** devuelve una sola tupla. Para obtener todas las tuplas del resultado, el programa debe incorporar un bucle para iterar sobre todas las tuplas. SQL incorporado ayuda a los programadores en el tratamiento de esta iteración. Aunque las relaciones sean conceptualmente conjuntos, las tuplas del resultado de las consultas se encuentran en un orden físico determinado. Cuando el programa ejecuta una sentencia **open** sobre un cursor, ese cursor pasa a apuntar a la primera tupla del resultado. Cada vez que se ejecuta una instrucción **fetch,** el cursor se actualiza para que apunte a la siguiente tupla del resultado. Cuando no quedan más tuplas por procesar, la variable SQLSTATE de SQLCA vale ´02000´ (que significa «sin datos»). Por tanto, se puede utilizar un bucle **while** (o su equivalente) para procesar todas las tuplas del resultado.

Se debe utilizar la sentencia **close** para indicar al sistema de bases de datos que borre la relación temporal que guardaba el resultado de la consulta. Para el ejemplo anterior, esta instrucción tiene la forma:

EXEC SQL **close** *c*;

Las expresiones de SQL incorporado para la modificación **(update, insert** y **delete)** de las bases de datos no devuelven ningún resultado. Por tanto son, de algún modo, más sencillas de expresar. Una solicitud de modificación de la base de datos tiene la forma:

EXEC SQL <cualquier **update, insert** o **delete válido**>;

En la expresión de SQL para la modificación de la base de datos pueden aparecer variables del lenguaje anfitrión precedidas de dos puntos. Si se produce un error en la ejecución de la instrucción, se establece un diagnóstico en SQLCA.

Las relaciones de las bases de datos también se pueden actualizar mediante cursores. Por ejemplo, si se desea sumar 100 al atributo *sueldo* de todos los profesores del departamento de Música, se puede declarar un cursor como:

EXEC SQL
declare *c* **cursor for select** *
from *profesor*
where *nombre_dept* = ´Música´
for update;

Después se itera por las tuplas ejecutando operaciones **fetch** sobre el cursor (como se mostró anteriormente) y, tras obtener cada tupla, se ejecuta el siguiente código:

EXEC SQL
update *profesor*
set *sueldo* = *sueldo* + 100
where current of *c*;

Las transacciones se pueden comprometer utilizando EXEC SQL COMMIT, o retroceder utilizando EXEC SQL ROLLBACK.

Las consultas en SQL incorporado habitualmente se definen cuando se escribe el programa. Existen raras situaciones en las que se necesita definir la consulta en tiempo de ejecución. Por ejemplo, una interfaz de una aplicación puede permitir al usuario especificar las condiciones de selección sobre uno o más atributos de una relación, y puede construir una cláusula **where** de la consulta de SQL en tiempo de ejecución, con las condiciones solo sobre los atributos que haya indicado el usuario. En estos casos, se puede construir una cadena de consulta y prepararla en tiempo de ejecución mediante la sentencia EXEC SQL PREPARE <nombre-consulta> FROM :<variable>, y se puede abrir un cursor sobre el nombre de la consulta.

SQLJ

SQL integrado en Java, denominado SQLJ, proporciona las mismas características que el resto de implementaciones de SQL integrado, pero utilizando una sintaxis diferente que se parece más a las características de Java, como los iteradores. Por ejemplo, SQLJ utiliza la sintaxis #sql en lugar de EXEC SQL y, en lugar de cursores, utiliza la interfaz iterator de Java para obtener los resultados de la consulta. Por tanto, el resultado de ejecutar una consulta es un iterador de Java y se puede utilizar el método next() de la interfaz iterator de Java para recorrer las tuplas resultado, como en el ejemplo anterior se usa **fetch** sobre el cursor. El iterador debe tener declarados atributos que coincidan con los atributos del resultado de la consulta en SQL. El siguiente fragmento de código muestra el uso de los iteradores:

```
#sql iterator deptInfoIter (uString nombre_dept, int sueldoMed);
deptInfoIter iter = null;

#sql iter = { select nombre_dept, avg(sueldo)
 from profesor
 group by nombre_dept };
while (iter.next()) {
 String nombreDept = iter.nombre_dept();
 int sueldoMed = iter.sueldoMed();
 System.out.println(nombreDept + " " + sueldoMed);
}
iter.close();
```

SQLJ lo soportan DB2 de IBM y Oracle; ambos proporcionan traductores para convertir código de SQLJ en código de JDBC. El traductor puede conectarse a la base de datos para comprobar la corrección sintáctica de las consultas en tiempo de compilación, y asegurar que los tipos de los resultados de las consultas con compatibles con los tipos de las variables a las que se asignan. Hasta 2009, SQLJ no lo soportaba ningún otro sistema de bases de datos.

Aquí no se describirá con detalle SQLJ; consulte las notas bibliográficas para obtener más información.

5.2. Funciones y procedimientos

Ya se han tratado distintas funciones que se incorporan al lenguaje SQL. En esta sección se muestra cómo los desarrolladores pueden escribir sus propias funciones y procedimientos, almacenarlos en la base de datos y, después, invocarlos desde sentencias de SQL. Las funciones resultan particularmente útiles con los tipos de datos especializados como las imágenes y los objetos geométricos. De hecho, un tipo de dato segmento utilizado en una base de datos de mapas puede tener una función asociada que comprueba si dos segmentos se superponen, y un tipo de datos imagen puede tener funciones asociadas para comparar las similitudes entre dos imágenes.

Las funciones y los procedimientos permiten almacenar la «lógica de negocio» en la base de datos y ejecutarla con sentencias de SQL. Por ejemplo, las universidades suelen tener muchas reglas sobre en cuántas asignaturas se puede matricular un estudiante en un semestre, el número mínimo de asignaturas que puede enseñar un profesor a tiempo completo, el máximo número de asignaturas en que se puede matricular un estudiante, etc. Aunque esta lógica de negocio se puede codificar en procedimientos del lenguaje de programación y almacenarlos totalmente aparte de la base de datos, definirlos y almacenarlos en la base de datos presenta algunas ventajas. Por ejemplo, permite que se puedan utilizar distintas aplicaciones y permite que exista un único punto de cambio si se modifica la lógica de negocio, sin modificar otras partes de la aplicación. El código de la aplicación puede invocar los procedimientos almacenados en lugar de actualizar directamente las relaciones de la base de datos.

SQL permite la definición de funciones, procedimientos y métodos. Se pueden definir utilizando un componente procedimental de SQL o un lenguaje de programación externo como Java, C, o C++. En primer lugar se van a tratar las definiciones en SQL y después se verá el uso de los lenguajes externos, en la Sección 5.2.3.

Aunque la sintaxis que se presenta viene definida en la norma de SQL, la mayoría de las bases de datos implementan versiones no estándar de esta sintaxis. Por ejemplo, el lenguaje procedimental de Oracle (PL/SQL), de SQL Server de Microsoft (TransactSQL) y de PostgreSQL (PL/pgSQL) son diferentes de la sintaxis estándar que se presenta aquí.

Más adelante se mostrarán estas diferencias para el caso de Oracle. Consulte los respectivos manuales del sistema para más información. Aunque algunas partes de la sintaxis que se presenta puede que no se puedan utilizar en estos sistemas, los conceptos que se describen son aplicables a todas las implementaciones, aunque con una sintaxis diferente.

5.2.1. Declaración e invocación de funciones y procedimientos de SQL

Suponga que se desea una función que, dado el nombre de un departamento, devuelva el número de profesores de dicho departamento.

Se puede definir la función como se muestra en la Figura 5.5.[4] Esta función puede utilizarse en una consulta que devuelva el nombre y el presupuesto de todos los departamentos con más de 12 profesores:

```
select nombre_dept, presupuesto
from profesor
where cuenta_dept(nombre_dept) > 12;
```

La norma SQL soporta funciones que pueden devolver tablas como resultado; esas funciones se denominan **funciones de tabla.**[5]

Considere la función definida en la Figura 5.6. Esa función devuelve una tabla que contiene todos los profesores de un determinado departamento. Observe que se hace referencia a los parámetros de las funciones anteponiendo el nombre de la función (*profesor_de.nombre_dept*).

La función puede emplearse en consultas de la manera siguiente:

```
select *
from table(profesor_de('Finanzas'));
```

Esta consulta devuelve todos los profesores del departamento de Finanzas. En el sencillo caso anterior, resulta fácil escribir esta consulta sin emplear funciones evaluadas sobre tablas. En general, sin embargo, las funciones evaluadas sobre tablas pueden considerarse **vistas parametrizadas** que generalizan el concepto habitual de las vistas permitiendo la introducción de parámetros.

SQL también permite el uso de procedimientos. La función *cuenta_dept* se puede escribir en forma de procedimiento:

```
create procedure cuenta_dept_proc
        (in nombre_dept varchar(20), out cuenta integer)
  begin
    select count(*) into cuenta
    from profesor
    where profesor.nombre_dept = cuenta_dept_proc.nombre_dept
  end
```

Las palabras clave **in** y **out** indican, respectivamente, parámetros que se espera que tengan valores asignados y parámetros cuyos valores se establecen en el procedimiento para devolver en ellos los resultados.

Se pueden invocar los procedimientos mediante la sentencia **call** desde otro procedimiento de SQL o desde SQL incorporado:

```
declare cuenta_de integer;
call cuenta_dept_proc('Física', cuenta_de);
```

Los procedimientos y las funciones pueden invocarse desde SQL dinámico, como se ilustra mediante la sintaxis de JDBC en la Sección 5.1.1.4.

SQL permite que haya más de un procedimiento con el mismo nombre, siempre que el número de argumentos de los procedimientos con el mismo nombre sea diferente. El nombre, junto con el número de argumentos, se utiliza para identificar los procedimientos. SQL permite también más de una función con el mismo nombre, siempre que las diferentes funciones con el mismo nombre tengan diferente número de argumentos o, para funciones con el mismo número de argumentos, se diferencien en el tipo de datos de un argumento, como mínimo.

```
create function cuenta_dept(nombre_dept varchar(20))
  returns integer
  begin
  declare cuenta integer;
    select count(*) into cuenta
    from profesor
    where profesor.nombre_dept = nombre_dept
  return cuenta;
  end
```

Figura 5.5. Función definida en SQL.

```
create function profesor_de(nombre_dept varchar(20))
  returns table (
    ID varchar (5),
    nombre varchar (20),
    nombre_dept varchar (20),
    sueldo numeric (8,2))
return table
  (select ID, nombre, nombre_dept, sueldo
  from profesor
  where profesor.nombre_dept = profesor_de.nombre_dept);
```

Figura 5.6. Función de tabla en SQL.

4 Si escribe sus propias funciones o procedimientos, debería escribir «**create or replace**» en lugar de **créate,** de forma que sea más sencillo modificar el código (sustituyendo la función) durante su depuración.

5 Esta característica apareció por primera vez en SQL:2003.

5.2.2. Construcciones del lenguaje para procedimientos y funciones

SQL soporta varias construcciones que le proporcionan casi toda la potencia de los lenguajes de programación de propósito general. La parte de la norma SQL que trata de estas construcciones se denomina **módulo de almacenamiento persistente** (Persistent Storage Module, **PSM**).

Las variables se declaran utilizando la sentencia **declare** y puede tener cualquier tipo válido de SQL. Las asignaciones se realizan mediante la sentencia **set.**

Las sentencias compuestas son de la forma **begin... end,** y pueden contener varias sentencias de SQL entre **begin** y **end.** En las sentencias compuestas pueden declararse variables locales, como se ha visto en la Sección 5.2.1. Una sentencia compuesta de la forma **begin atomic... end** asegura que todas las sentencias se ejecutan como una única transacción.

SQL:1999 soporta las sentencias **while** y **repeat** con la sintaxis siguiente:

```
while expresión_booleana do
    secuencia de sentencias;
end while
repeat
    secuencia de sentencias;
until expresión_booleana
end repeat
```

También hay un bucle **for** que permite la iteración por todos los resultados de una consulta:

```
declare n integer default 0;
for r as
        select presupuesto from departamento
        where nombre_dept = 'Música'
do
      set n = n− r.presupuesto
end for
```

El programa obtiene los valores fila a fila en la variable de bucle **for** (r, en este ejemplo). Se puede emplear la sentencia **leave** para salir del bucle, mientras que **iterate** comienza con la siguiente tupla, a partir del comienzo del bucle, y se salta las instrucciones restantes.

Entre las instrucciones condicionales soportadas por SQL están las instrucciones **if-then-else,** con la sintaxis:

```
if expresión_booleana
    then sentencia o sentencias compuestas
elseif expresión_booleana
    then sentencia o sentencias compuestas
else sentencia o sentencias compuestas
end if
```

SQL también soporta sentencias **case** parecidas a la sentencia case del lenguaje C/C++ (además de las expresiones case, que se vieron en el Capítulo 3).

La Figura 5.7 proporciona un ejemplo de mayor tamaño del uso de construcciones procedimentales en SQL. El procedimiento *registrarEstudiante* definido en la figura registra un estudiante en una sección de una asignatura, tras verificar que el número de estudiantes en la sección no supera la capacidad del aula asignada a dicha sección. La función devuelve un código de error; si tiene un valor mayor o igual a cero indica que se ha realizado, mientras que un valor negativo indica una condición de error y se devuelve un mensaje indicando la razón del fallo en el parámetro **out.**

El lenguaje procedimental de SQL también soporta la señalización de las **condiciones de excepción** y la declaración de **manejadores** que pueden tratar esa excepción, como en este código:

```
declare sin_sitio_en_aula condition
declare exit handler for sin_sitio_en_aula
begin
secuencia de sentencias
end
```

Las sentencias entre **begin** y **end** pueden provocar una excepción ejecutando **signal** *sin_sitio_en_aula*. El manejador dice que si se da la condición, la acción que hay que tomar es salir de la sentencia **begin end** circundante. Una acción alternativa sería **continue,** que continúa con la ejecución a partir de la siguiente instrucción que ha generado la excepción. Además de las condiciones definidas de manera explícita, también hay condiciones predefinidas como **sqlexception, sqlwarning** y **not found.**

```
– – Registra un estudiante tras asegurarse de que no se excede
    la capacidad del aula.
– – Devuelve 0 si va bien, y -1 si se excede la capacidad.
create function registraEstudiante (
  in e_id varchar(5),
  in e_asignaturaid varchar (8),
  in e_seccid varchar (8),
  in e_semestre varchar (6),
  in e_año numeric (4,0),
  out errorMensg varchar(100)
returns integer
begin
  declare actualEnrol int;
  select count(*) into actualEnrol
    from matricula
    where asignatura_id = e_asignaturaid and secc_id = e_seccid
          and semestre = e_semestre and año = e_año;
  declare límite int;
  select capacidad into límite
    from aula natural join sección
    where asignatua_id = e_asignaturaid and secc_id = e_seccid
          and semestre = e_semestre and año = e_año;
  if (actualEnrol < límite)
    begin
      insert into matricula values
            (e_id, e_asignaturaid, e_seccid, e_semestre, e_año, null);
      return(0);
    end
  – –De lo contario, ya se ha llegado al límite de capacidad de la sección
  set errorMensg = 'Límite de matricula alcanzado para la asignatura
      ' || e_asignaturaid || ' seccion ' || e_seccid;
  return(-1);
end;
```

Figura 5.7. Procedimiento para registrar un estudiante en una sección de una asignatura.

5.2.3. Rutinas en otros lenguajes

Aunque las extensiones procedimentales a SQL pueden resultar muy útiles, desafortunadamente no se soportan de manera estándar por distintas bases de datos. Incluso las características más básicas tienen una sintaxis o una semántica diferentes según las distintas bases de datos. En consecuencia, los programadores tienen que aprender un nuevo lenguaje para cada producto de base de datos. Una alternativa que está ganando adeptos es definir los procedimientos en un lenguaje imperativo, pero que se pueda llamar desde consultas de SQL, y definir disparadores.

SQL permite definir funciones en lenguajes de programación como Java, C#, C o C++. Las funciones definidas de esta manera pueden ser más eficientes que las definidas en SQL, y los cálculos que no pueden llevarse a cabo en SQL pueden ejecutarse mediante estas funciones.

Los procedimientos externos y las funciones externas pueden especificarse de esta manera (tenga en cuenta que la sintaxis concreta depende del sistema de base de datos concreto que se utilice):

create procedure cuenta_dept_proc(**in** *nombre_dept* **varchar**(20),
out count **integer**)
language C
external name ´/usr/avi/bin/cuenta_dept_proc
create function cuenta_dept(*nombre_dept* **varchar**(20))
returns integer
language C
external name ´/usr/avi/bin/ cuenta_dept

En general, los procedimientos del lenguaje externo tienen que tratar con valores nulos (tanto **in** como **out**) y con valores de retorno. También necesitan comunicar el estado de éxito/error, para tratar con las excepciones. Esta información se puede comunicar mediante parámetros extra: un valor **sqlstate** para indicar el estado de éxito/fracaso, un parámetro para guardar el valor devuelto por la función y variables indicadoras para el resultado de cada parámetro o función, para indicar si su valor es nulo. También se pueden utilizar otros mecanismos para manejar los valores null, por ejemplo pasando punteros en lugar del valor; el mecanismo exacto depende de la base de datos. Sin embargo, si una función no trata con estas situaciones, se puede añadir una línea extra **parameter style general** a la declaración anterior indicando que los procedimientos externos o las funciones externas solo toman los argumentos mostrados y no tratan con valores nulos ni excepciones.

Las funciones definidas en un lenguaje de programación y compiladas fuera del sistema de base de datos pueden cargarse y ejecutarse con el código del sistema de bases de datos. Sin embargo, ello conlleva el riesgo de que un fallo del programa corrompa las estructuras internas de la base de datos y pueda saltarse la funcionalidad de control de accesos del sistema de bases de datos. Los sistemas de bases de datos que se preocupan más del rendimiento eficiente que de la seguridad pueden ejecutar los procedimientos de este modo. Los sistemas de bases de datos que están preocupados por la seguridad pueden ejecutar ese código como parte de un proceso diferente, comunicarle el valor de los parámetros y recuperar los resultados mediante la comunicación entre procesos. Sin embargo, la sobrecarga de tiempo que supone la comunicación entre procesos es bastante elevada; en las arquitecturas de CPU habituales, decenas a centenares de miles de instrucciones se pueden ejecutar en el tiempo necesario para una comunicación entre los procesos.

Si el código está escrito en un lenguaje «seguro» como Java o C#, cabe otra posibilidad: ejecutar el código en un **arenero** (sandbox) dentro del propio proceso de ejecución de la consulta a la base de datos. El arenero permite que el código de Java o de C# tenga acceso a su propia área de memoria, pero evita que ese código lea o actualice la memoria del proceso de ejecución de la consulta, o tenga acceso a los archivos del sistema de archivos (la creación de areneros no es posible en lenguajes como C, que permite el acceso sin restricciones a la memoria mediante los punteros). Evitar la comunicación entre procesos reduce enormemente la sobrecarga de llamadas a funciones.

Varios sistemas de bases de datos actuales soportan que las rutinas de lenguajes externos se ejecuten en areneros dentro del proceso de ejecución de las consultas. Por ejemplo, Oracle y DB2 de IBM permiten que las funciones de Java se ejecuten como parte del proceso de la base de datos. SQL Server de Microsoft permite que los procedimientos compilados en CLR (Common Language Runtime) se ejecuten dentro del proceso de la base de datos; esos procedimientos pueden haberse escrito, por ejemplo, en C# o en Visual Basic. PostgreSQL permite funciones definidas en distintos lenguajes, como Perl, Python y Tcl.

5.3. Disparadores

Un **disparador** es una sentencia que el sistema ejecuta de manera automática como efecto secundario de la modificación de la base de datos. Para diseñar un mecanismo disparador se deben cumplir dos requisitos:

1. Especificar las condiciones en las que se va a ejecutar el disparador. Esto se descompone en un *evento* que provoca la comprobación del disparador y una *condición* que se debe cumplir para que se ejecute el disparador.
2. Especificar las *acciones* que se van a realizar cuando se ejecute el disparador.

Una vez que se introduce un disparador en la base de datos, el sistema de bases de datos asume la responsabilidad de ejecutarlo cada vez que se produzca el evento especificado y se satisfaga la condición correspondiente.

SINTAXIS NO ESTÁNDAR PARA PROCEDIMIENTOS Y FUNCIONES

Aunque la norma de SQL define la sintaxis de los procedimientos y funciones, la mayoría de los sistemas de bases de datos no siguen estrictamente la norma, y existe una variación considerable en la sintaxis que admiten. Una de las razones de esta situación es que estas bases de datos introdujeron el soporte para procedimientos y funciones antes de que se normalizase la sintaxis y continuaron soportando la sintaxis original. No es posible listar la sintaxis de todas las bases de datos, pero se mostrarán algunas de las pequeñas diferencias en el caso de PL/SQL de Oracle, mediante una versión de la función de la Figura 5.5, como se definiría en PL/SQL.

Aunque las dos versiones son similares en concepto, existen un cierto número de pequeñas diferencias sintácticas, algunas de ellas evidentes cuando se comparan ambas versiones de la función. Aunque no se muestra aquí, la sintaxis para el control de flujo en PL/SQL también tiene algunas diferencias con la sintaxis presentada.

Observe que PL/SQL permite que se especifique un tipo como tipo de un atributo de una relación, añadiendo el sufijo *%type*. Por otra parte, PL/SQL no soporta directamente la capacidad de devolver una tabla, aunque existe una forma indirecta de implementar esta funcionalidad creando un tipo tabla. Los lenguajes procedimentales que soportan otras bases de datos también tienen cierto número de diferencias sintácticas. Consulte las respectivas referencias del lenguaje para más información.

create or replace function *cuenta_dept*(*nombre_dept*
in *profesor.nombre_dept %type*)
return integer
as
cuenta_de **integer**;
begin
select count(*) **into** *cuenta_de*
from *profesor*
where *profesor.nombre_dept* = *nombre_dept*;
return *cuenta_de*;
end;

5.3.1. Necesidad de los disparadores

Los disparadores se pueden utilizar para implementar ciertas restricciones de integridad que no se pueden especificar utilizando el mecanismo de restricciones de SQL. Los disparadores también son mecanismos útiles para alertar a los usuarios o para iniciar de manera automática ciertas tareas cuando se cumplen determinadas condiciones. A modo de ejemplo, se puede diseñar un disparador de forma que siempre que se inserte una tupla en la relación *matricula*, se actualice la tupla de la relación *estudiante* del estudiante que se matricula en la asignatura añadiendo el número de créditos de dicha asignatura al número total de créditos del estudiante. Otro ejemplo, suponga que un almacén desea mantener un inventario mínimo de cada elemento; cuando el nivel de inventario de un elemento cae por debajo del nivel mínimo, se puede crear una petición automáticamente. Cuando se realiza una actualización de nivel de inventario de un elemento, el disparador compara el nivel de inventario actual con el nivel mínimo de ese elemento, y si el nivel se encuentra en el mínimo o por debajo, se crea un nuevo pedido.

Observe que los sistemas de disparadores en general no pueden realizar actualizaciones fuera de la base de datos y, por tanto, en el ejemplo de los pedidos del inventario no se puede usar un disparador para realizar directamente un pedido al exterior. En su lugar se añade un pedido a la relación de pedidos. Se debe crear un proceso del sistema separado que se esté ejecutando constantemente y que explore periódicamente la relación pedidos y formule los pedidos. Algunos sistemas de bases de datos ofrecen soporte predeterminado para el envío de correo electrónico desde las consultas y los disparadores de SQL usando el enfoque mencionado.

5.3.2. Los disparadores en SQL

Ahora se tratará de establecer cómo implementar los disparadores en SQL. La sintaxis que se presenta es la definida por la norma de SQL, pero la mayoría de las bases de datos implementan versiones no estándar de esta sintaxis. Aunque la sintaxis que se presenta no la soporten la mayoría de esos sistemas, los conceptos que se describen son aplicables a todas las implementaciones. Más adelante en esta sección se tratarán las implementaciones no estándar de los disparadores.

La Figura 5.8 muestra cómo se pueden utilizar los disparadores para forzar la integridad referencial sobre el atributo *franja_horaria_id* de la relación *sección*. La primera definición de disparador de la figura especifica que el disparador se inicia *después* de cualquier inserción en la relación *sección* y asegura que el valor de *franja_horaria_id* que se inserta es válido. Cada sentencia de inserción de SQL puede insertar varias tuplas de la relación, y la cláusula **for each row** del código del disparador se podría iterar luego de manera explícita para cada fila insertada. La cláusula **referencing new row as** crea una variable *fila_nueva* (denominada **transition variable**), que almacena el valor de la fila insertada después de la inserción.

La sentencia **when** especifica una condición. El sistema ejecuta el resto del cuerpo del disparador solo para las tuplas que satisfacen la condición. La cláusula **begin atomic... end** sirve para reunir varias sentencias de SQL en una sola sentencia compuesta. En el ejemplo, solo existe una única sentencia, que retrocede la transacción que generó que se ejecutase el disparador. Por tanto, cualquier transacción que viola la restricción de integridad referencial se retrocede, asegurando que los datos de la base de datos satisfacen la restricción.

No es suficiente comprobar la integridad referencial solo en las inserciones, también hay que considerar las actualizaciones de *sección*, así como los borrados y actualizaciones de la tabla *franja_horaria*. Este disparador comprueba que el *franja_horaria_id* de la tupla que se desea borrar aun está presente en *franja_horaria*, o que ninguna tupla de *sección* contiene ese valor de *franja_horaria_id*; si no fuera así, la integridad referencial no se cumpliría.

Para asegurar la integridad referencial, se deberían crear también disparadores que manejasen las actualizaciones de *sección* y *franja_horaria*; a continuación se describe cómo se pueden ejecutar disparadores en las actualizaciones, pero se deja la definición de los mismos como un ejercicio para el lector.

Para las actualizaciones, el disparador puede especificar los atributos cuya actualización provoca su ejecución; las actualizaciones de otros atributos no causarán que se ejecute. Por ejemplo, para especificar que el disparador se ejecuta tras una actualización del atributo *nota* de la relación *matricula*, se escribe:

after update of *matricula* **on** *nota*

La cláusula **referencing old row as** se puede utilizar para crear una variable que almacene el valor anterior de una fila actualizada o borrada. La cláusula **referencing new row as** se puede usar con las actualizaciones además de con las inserciones.

La Figura 5.9 muestra cómo se puede utilizar un disparador para mantener actualizado el valor del atributo *tot_créditos* de las tuplas *estudiante* cuando se actualiza el atributo *nota* de una tupla de la relación *matricula*. El disparador se ejecuta solo si el atributo *nota* se actualiza con un valor distinto de *null* o 'F', con un valor que indica que la asignatura se ha superado. La sentencia **update** sigue la sintaxis normal de SQL excepto por su uso de la variable *fila_nueva*.

Una implementación más realista de este ejemplo de disparador también manejaría la corrección de notas que modificarán el estatus de superación de una asignatura frente a su no superación, así como las inserciones en la relación *matricula* en las que el valor de *nota* indica la superación de la asignatura. Se deja como ejercicio para el lector.

Como ejemplo adicional de uso de un disparador, la acción **delete** de una tupla *estudiante* se podría utilizar para comprobar si el estudiante tiene alguna entrada en la relación *matricula* y, si es así, eliminarla.

Muchos sistemas de bases de datos soportan otros muchos eventos disparadores, como puede ser cuando un usuario (o aplicación) se registra en la base de datos (es decir, abre una conexión), si el sistema se apaga o si se realizan cambios en la configuración del sistema.

Los disparadores se pueden activar antes **(before)** del evento **(insert, delete** o **update)** en lugar de después **(after)** del evento. Los disparadores que se ejecutan antes del evento pueden servir como restricciones adicionales que pueden impedir actualizaciones, inserciones o borrados no válidos. En lugar de dejar que la acción no válida proceda y genere un error, el disparador podría tomar acciones para corregir el problema de forma que la actualización, inserción o borrado fuesen válidos. Por ejemplo, si se intenta insertar un profesor en un departamento cuyo nombre no aparece en la relación *departamento*, el disparador podría insertar una tupla en la relación *departamento* con ese nombre de departamento antes de que la inserción generase una violación de clave externa. Como ejemplo adicional, suponga que el valor de una nota a insertar quedase en blanco, supuestamente para indicar la ausencia de nota. Se puede definir un disparador que sustituye el valor por un valor **null.** Se puede utilizar la sentencia **set** para realizar estas modificaciones. Un ejemplo de este tipo de disparador aparece en la Figura 5.10.

En lugar de llevar a cabo una acción por cada fila afectada, se puede realizar una sola acción para toda la sentencia de SQL que ha causado la inserción, el borrado o la actualización. Para hacerlo se utiliza la cláusula **for each statement** en lugar de **for each row.** Las cláusulas **referencing old table as** o **referencing new table as** se pueden emplear para hacer referencia a tablas temporales (denominadas *tablas de transición*) que contengan todas las filas afectadas. Las tablas de transición no se pueden emplear con los

```
create trigger franja_horaria_comprobacion1 after insert on sección
referencing new row as fila_nueva
for each row
when (fila_nueva.franja_horaria_id not in (
        select franja_horaria_id
        from franja_horaria)) /* franja_horaria_id no está en franja_horaria*/
begin
   rollback
end;
create trigger franja_horaria_comprobacion2 after delete on franja_horaria
referencing old row as fila_ant
for each row
when (fila_ant.franja_horaria_id not in (
        select franja_horaria_id
        from franja_horaria) /* última tupla de franja_horaria_id
        borrada de franja_horaria*/
and fila_ant.franja_horaria_id in (
         select franja_horaria_id
         from sección)) /* y franja_horaria_id aún con referencias
         en sección*/
begin
   rollback
end;
```

Figura 5.8. Utilizando disparadores para mantener la integridad referencial.

```
create trigger créditos_obtenidos after update of matricula on (nota)
referencing new row as fila_nueva
referencing old row as fila_ant
for each row
when fila_nueva.nota <> ´F´ and fila_nueva.nota is not null
   and (fila_ant.nota = ´F´ or fila_ant.nota is null)
begin atomic
   update estudiante
set tot_créditos = tot_créditos+
                  (select créditos
                   from asignatura
                   where asignatura.asignatura_id= fila_nueva.asignatura_id)
   where estudiante.id = fila_nueva.id;
end;
```

Figura 5.9. Uso de un disparador para mantener los valores de *créditos_obtenidos*.

```
create trigger setnull before update on matricula
referencing new row as fila_nueva
for each row
when (fila_nueva.nota = ´ ´)
begin atomic
   set fila_nueva.nota = null;
end;
```

Figura 5.10. Ejemplo de uso de **set** para modificar un valor insertado.

```
create trigger nuevo_pedido after update of cantidad on inventario
referencing old row as fila_ant, new row as fila_nueva
for each row
when fila_nueva.nivel <= (select nivel
                          from minnivel
                          where minnivel.producto = fila_ant.producto)
and fila_ant.nivel> (select nivel
                     from minnivel
                     where minnivel.producto = fila_ant.producto)
begin atomic
   insert into pedidos (select producto, cantidad
                        from nuevo_pedido
                        where nuevo_pedido.producto = fila_ant.producto);
end;
```

Figura 5.11. Ejemplo de disparador para realizar un nuevo pedido de un producto.

disparadores **before,** pero sí con los disparadores **after,** independientemente de si son disparadores de sentencias (statement) o de filas (row). Se puede utilizar una única sentencia de SQL para llevar a cabo varias acciones con base en las tablas de transición.

Los disparadores se pueden habilitar y deshabilitar; de manera predeterminada, al crearlos se habilitan, pero se pueden deshabilitar empleando **alter trigger** *nombre_disparador* **disable** (algunas bases de datos utilizan una sintaxis alternativa, como **disable trigger** *nombre_disparador*). Los disparadores deshabilitados se pueden volver a habilitar. Los disparadores también se pueden eliminar, lo que los elimina de manera permanente, usando la instrucción **drop trigger** *nombre_disparador*.

Volviendo al ejemplo del inventario del almacén, suponga que se tienen las siguientes relaciones:

- *inventario*(*producto*, *nivel*), que indica la cantidad de producto actualmente en el almacén.
- *minnivel* (*producto*, *nivel*), que indica la cantidad mínima que se debe mantener de cada producto.
- *nuevo_pedido*(*producto*, *cantidad*), que indica la cantidad de producto que se debe pedir cuando su nivel cae por debajo del mínimo.
- *pedidos*(*producto*, *cantidad*), que indica la cantidad de producto que se debe pedir.

Obsérvese que se ha tenido mucho cuidado a la hora de realizar los pedidos solo cuando su cantidad cae desde un nivel por encima del mínimo a un nivel por debajo del mínimo. Si solo se comprueba si el nuevo valor después de la actualización está por debajo del mínimo, se puede realizar erróneamente un pedido cuando ya se ha realizado. Se puede usar el disparador mostrado en la Figura 5.11 para volver a pedir el producto.

Los sistemas de bases de datos basados en SQL utilizan ampliamente los disparadores, aunque antes de SQL:1999 no formaban parte de la norma de SQL. Desafortunadamente, cada sistema de bases de datos implementa su propia sintaxis para los disparadores, lo que conlleva incompatibilidades. La sintaxis para los disparadores de SQL:1999 que se ha utilizado es similar, pero no idéntica, a la sintaxis de los sistemas de bases de datos DB2 de IBM y de Oracle.

5.3.3. Cuándo no deben emplearse los disparadores

Existen muchas aplicaciones para los disparadores, como las que se acaban de ver en la Sección 5.3.2, pero algunas se manejan mejor con técnicas alternativas. Por ejemplo, se puede implementar la característica **on delete cascade** de una restricción de clave externa mediante un disparador, en lugar de la característica en cascada. No solo implicaría un mayor trabajo de implementación, también sería mucho más difícil para el usuario de la base de datos entender el conjunto de restricciones que implementa la base de datos.

Como ejemplo adicional, se pueden utilizar los disparadores para mantener las vistas materializadas. De hecho si se desea un acceso muy rápido al número total de estudiantes registrados en cada sección de una asignatura, se podría realizar creando una relación:

registro_sección (*asignatura_id*, *seec_id*, *semestre*, *año*, *total_estudiantes*)

definida por la consulta:

```
select asignatura_id, secc_id, semestre, año,
       count(ID) as total_estudiantes
from matricula
group by asignatura_id, secc_id, semestre, año;
```

SINTAXIS NO ESTÁNDAR DE LOS DISPARADORES

Aunque la sintaxis descrita de los disparadores forma parte de la norma de SQL, y la soporta DB2 de IBM, la mayoría de los sistemas de bases de datos siguen una sintaxis no estándar para especificar los disparadores, y puede que no implementen todas las características de la norma SQL. A continuación se describen algunas de las diferencias; consulte los respectivos manuales del sistema para obtener más información.

Por ejemplo, en la sintaxis de Oracle, en lugar de la sintaxis estándar de SQL, la palabra clave **row** no aparece en la sentencia **referencing.** La palabra clave **atomic** no aparece tras **begin.** La referencia a *fila_nueva* en la sentencia **select** anidada en la sentencia **update** debe empezar con dos puntos para indicar al sistema que la variable *fila_nueva* se ha definido externamente a la sentencia de SQL. Además, no se permiten subconsultas en las cláusulas **when** ni **if.** Es posible solventar este problema trasladando predicados complejos de la cláusula **when** a una consulta separada y guardar el resultado en una variable local y, posteriormente, referenciar esta variable en una cláusula **if,** trasladando el cuerpo del disparador a la cláusula **then** correspondiente. Además, en Oracle no se permite que los disparadores ejecuten directamente el retroceso de una transacción; sin embargo, pueden utilizar una función llamada raise_application_error no solo para retroceder la transacción, sino también para devolver un mensaje de error al usuario/aplicación que realiza la actualización.

Como ejemplo adicional, en SQL Server de Microsoft se utiliza la palabra clave **on** en lugar de **after.** La cláusula **referenciada** se omite, y las filas nueva y anterior se referencian con las variables de tupla **deleted** e **inserted.** Además, la cláusula **for each row** se omite, y se sustituye **when** por **if.** La especificación de **before** no está soportada, pero se admite una especificación **instead of.**

Los disparadores de PostgreSQL no tienen cuerpo, pero en su lugar invocan a un procedimiento para cada fila, que puede acceder a las variables **new** y **old** que contienen los valores anterior y nuevo de las filas. En lugar de realizar un retroceso, el disparador puede lanzar una excepción, con un mensaje de error asociado.

Hay que mantener actualizado el valor de *total_estudiantes* para cada asignatura mediante los disparadores de inserción, borrado o actualizado de la relación *matricula*. Este mantenimiento puede requerir la inserción, actualización o borrado de tuplas de *sección_registro* y hay que escribir los disparadores de la forma adecuada.

Sin embargo, muchos sistemas de bases de datos actuales soportan las vistas materializadas que mantiene el sistema de bases de datos (consulte la Sección 4.2.3). En consecuencia, no hay necesidad de escribir código de disparadores para mantener estas vistas materializadas.

Los disparadores también se han usado para manipular copias, o réplicas, de las bases de datos. Se puede crear una colección de disparadores sobre la inserción, borrado o actualización de cada relación para registrar las modificaciones en relaciones denominadas **cambio** o **delta.** Un proceso diferente copiaba las modificaciones a la réplica de la base de datos. Los sistemas de bases de datos modernos proporcionan características predeterminadas para la réplica de las bases de datos, lo que hace que en la mayor parte de los casos los disparadores resulten innecesarios para las réplicas. La réplica de bases de datos se trata en detalle en el Capítulo 19.

Otro problema con los disparadores es la ejecución no pretendida de la acción disparada cuando se cargan datos de una copia de seguridad,[6] o cuando las actualizaciones de la base de datos de un sitio se replican en un sitio de copia de seguridad. En esos casos, la acción del disparador ya se ha ejecutado y, generalmente, no se debe volver a ejecutar. Al cargar los datos, los disparadores se pueden deshabilitar de manera explícita. Para los sistemas de réplica de copia de seguridad que puede que tengan que sustituir el sistema principal, hay que deshabilitar los disparadores en un principio y habilitarlos cuando el sitio de copia de seguridad sustituye al principal en el procesamiento. Como alternativa, algunos sistemas de bases de datos permiten que los disparadores se especifiquen como **not for replication** (no para réplica), lo que garantiza que no se ejecuten en el sitio de copia de seguridad durante la réplica de la base de datos. Otros sistemas de bases de datos ofrecen una variable del sistema que denota que la base de datos es una réplica en la que se están repitiendo las acciones de la base de datos; el cuerpo del disparador debe examinar esta variable y salir si es verdadera. Ambas soluciones eliminan la necesidad de habilitar y deshabilitar de manera explícita los disparadores.

6 Se tratan con detalle la copia de seguridad y la recuperación frente a fallos en el Capítulo 16.

Los disparadores se deben escribir con sumo cuidado, dado que un error en un disparador detectado en tiempo de ejecución provoca el fallo de la instrucción de inserción, borrado o actualización que inició el disparador. Además, la acción de un disparador puede activar otro disparador. En el peor de los casos, esto puede dar lugar a una cadena infinita de disparos. Por ejemplo, suponga que un disparador de inserción sobre una relación tiene una acción que provoca otra (nueva) inserción sobre la misma relación. La acción de inserción dispara otra acción de inserción, y así hasta el infinito. Algunos sistemas de bases de datos limitan la longitud de las cadenas de disparadores (por ejemplo, a 16 o 32), y consideran que las cadenas de disparadores de mayor longitud son un error. Otros sistemas dan una indicación de error a cualquier disparador que intenta referenciar la relación cuya modificación genera que se ejecute el disparador en primer lugar.

Los disparadores pueden utilizarse para objetivos muy útiles, pero es preferible evitarlos si existen alternativas. Muchas aplicaciones de los disparadores se pueden sustituir por el uso apropiado de los procedimientos almacenados, que se trataron en la Sección 5.2.

5.4. Consultas recursivas **

Considere el ejemplar de la relación *prerreq* que se muestra en la Figura 5.12, que contiene la información sobre las distintas asignaturas que ofrece la universidad y los prerrequisitos de cada asignatura.[7]

Suponga ahora que se desea encontrar qué asignaturas son un prerrequisito directa o indirectamente, para una asignatura dada, digamos CS-347. Es decir, se desean encontrar las asignaturas que son prerrequisito directo de CS-347, o es prerrequisito de una asignatura que es prerrequisito de CS-347, y así sucesivamente.

asignatura_id	*prerreq_id*
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Figura 5.12. La relación *prerreq*.

7 Este ejemplar de *prerreq* difiere del utilizado anteriormente por razones que se verán cuando se utilice para explicar las consultas recursivas.

Por tanto, si CS-301 es un prerrequisito de CS-347, y CS-201 es un prerrequisito de CS-301, y CS-101 es un prerrequisito de CS-201, entonces CS-301, CS-201 y CS-101 son todas prerrequisitos de CS-347.

El **cierre transitivo** de la relación *prerreq* es una relación que contiene todos los pares (*aid*, *pre*) tal que *pre* es un prerrequisito directo o indirecto de *aid*. Existen numerosas aplicaciones que requieren el cálculo de cierres transitivos similares o **jerarquías.** Por ejemplo, las organizaciones suelen constar de varios niveles de unidades organizativas. Las máquinas constan de parte que tienen, a su vez, subpartes, etc; por ejemplo, una bicicleta puede tener subpartes como las ruedas y los pedales, que a su vez tienen subpartes, como las llantas y los radios. El cierre transitivo se puede utilizar en esas jerarquías para encontrar, por ejemplo, todas las partes de una bicicleta.

5.4.1. Cierre transitivo mediante iteración

Una manera de escribir la consulta anterior es emplear la iteración: en primer lugar hay que determinar las asignaturas que son prerrequisitos directos de CS-347, luego las que son prerrequisitos de todas las asignaturas bajo el primer conjunto, etc. Este proceso iterativo continúa hasta que se alcanza una iteración y no se añaden más asignaturas.

La Figura 5.13 muestra la función *buscarTodosReq*(*aid*) para llevar a cabo esa tarea; la función tiene como parámetro el *asig_id* de la asignatura (*aid*), calcula el conjunto de todos los prerrequisitos directos e indirectos de esa asignatura y devuelve el conjunto.

El procedimiento utiliza tres tablas temporales:

- *a_prerreq*: que almacena el conjunto de tuplas que se va a devolver.
- *nueva_a_prerreq*: que almacena las asignaturas localizadas en la iteración anterior.
- *temp*: que se utiliza como almacenamiento temporal mientras se manipulan los conjuntos de asignaturas.

Observe que SQL permite la creación de tablas temporales utilizando el comando **create temporary table;** estas tablas solo están disponibles dentro de la transacción que ejecuta la consulta, y se eliminan cuando la transacción termina. Más aún, si dos instancias de *buscarTodosReq* se ejecutan concurrentemente, cada una tendrá su propia copia de las tablas temporales; si comparten una copia, sus resultados serían incorrectos.

El procedimiento inserta todos los prerrequisitos directos de la asignatura *aid* en *nueva_a_prerreq* antes del bucle **repeat.** El bucle **repeat** añade primero todas las asignaturas de *nueva_a_prerreq* a *a_prerreq*. Luego calcula los prerrequisitos de todas las asignaturas en *nueva_a_prerreq*, excepto las que ya se sabe que son prerrequisito de *aid*, y las almacena en la tabla temporal *temp*. Finalmente, sustituye el contenido de *nueva_a_prerreq* por el de *temp*. El bucle **repeat** termina cuando no encuentra más prerrequisitos (indirectos) nuevos.

La Figura 5.14 muestra los prerrequisitos que se encontrarían en cada iteración si el procedimiento se llamara para la asignatura CS-347.

Hay que destacar que el uso de la cláusula **except** en la función garantiza que esta trabaje incluso en el caso (anormal) de que haya un ciclo de prerrequisitos. Por ejemplo, si *a* es un prerrequisito de *b*, *b* es un prerrequisito de *c* y *c* es un prerrequisito de *a*, existe un ciclo.

Aunque los ciclos pueden resultar poco realistas en los prerrequisitos de asignaturas, son posibles en otras aplicaciones. Por ejemplo, suponga que se tiene la relación *vuelos*(*a*, *desde*) que indica las ciudades a las que se puede llegar desde otra ciudad mediante un vuelo directo. Se puede escribir un código parecido al de la función *buscarTodosReq* para determinar todas las ciudades a las que se puede llegar mediante una serie de uno o más vuelos desde una ciudad dada. Todo lo que hay que hacer es sustituir *prerreq* por *vuelo* y el nombre de los atributos de manera acorde. En esta situación sí que puede haber ciclos de alcanzabilidad, pero la función trabajaría correctamente, ya que eliminaría las ciudades que ya se hubieran tomado en cuenta.

```
create function buscarTodosReq(aid varchar(8))
  – – Encuentra todas las asignaturas que son prerrequisitos (directos
      o indirectos) de aid.
returns table (asignatura_id varchar(8))
  – – La relación prerreq(asignatura_id, prerreq_id) especifica qué asig-
      natura es un prerrequisito directo de otra.
begin
  create temporary table a_prerreq(asignatura_id varchar(8));
    – – La tabla a_prerreq almacena el conjunto de asignaturas a
        devolver.
  create temporary table nueva_a_prerreq (asignatura_id varchar(8));
    – – La tabla nueva_a_prerreq contiene las asignaturas encontradas
        en la iteración anterior.
  create temporary table temp(asignatura_id varchar(8));
    – – La tabla temp se usa para guardar los resultados intermedios.
  insert into nueva_a_prerreq
    select prereq_id
    from prereq
    where asignatura_id = aid;
  repeat
    insert into a_prerreq
      select asignatura_id
      from nueva_a_prerreq;
    insert into temp
      (select prereq.asignatura_id
        from nueva_a_prerreq, prereq
        where nueva_a_prerreq.asignatura_id = prereq.prereq_id
      )
      except (
        select asignatura_id
        from a_prerreq
      );
    delete from nueva_a_prerreq;
    insert into nueva_a_prerreq
      select *
      from temp;
    delete from temp;
  until not exists (select * from nueva_a_prerreq)
  end repeat;
  return table a_prerreq;
end
```

Figura 5.13. Encontrar todos los prerrequisitos de una asignatura.

Número de iteración	Tuplas de a
0	
1	(CS-301)
2	(CS-301), (CS-201)
3	(CS-301), (CS-201)
4	(CS-301), (CS-201), (CS-101)
5	(CS-301), (CS-201), (CS-101)

Figura 5.14. Prerrequisitos de CS-347 en iteraciones de la función *buscarTodosReq*.

5.4.2. Recursión en SQL

Resulta muy poco práctico especificar el cierre transitivo empleando la iteración. Hay un enfoque alternativo, usando definiciones recursivas de las vistas, que resulta más sencillo de utilizar.

Se puede emplear la recursión para definir el conjunto de asignaturas que son prerrequisitos de una asignatura dada, digamos CS-347, del siguiente modo. Las asignaturas que son prerrequisitos (directa o indirectamente) de CS-347 son:

1. Asignaturas que son prerrequisitos de CS-347.
2. Asignaturas que son prerrequisitos de aquellas asignaturas que son prerrequisitos (directa o indirectamente) de CS-347.

Observe que el caso 2 es recursivo, ya que define el conjunto de asignaturas que son prerrequisitos de CS-347 en términos del conjunto de asignaturas que son prerrequisitos de CS-347. Otros ejemplos de cierre transitivo, como la búsqueda de todas las subpartes (directas o indirectas) de un componente dado, también pueden definirse de manera parecida, recursivamente.

Desde la versión SQL:1999, la norma de SQL soporta una forma limitada de recursión, que emplea la cláusula **with recursive,** en la que las vistas (o las vistas temporales) se expresan en términos de sí mismas. Se pueden utilizar consultas recursivas, por ejemplo, para expresar de manera concisa el cierre transitivo. Recuerde que la cláusula **with** se emplea para definir una vista temporal cuya definición solo está disponible para la consulta en la que se define. La palabra clave adicional **recursive** especifica que la vista es recursiva.

Por ejemplo, se pueden buscar todos los pares (*aid*, *pre*) tales que *pre* sea directa o indirectamente un prerrequisito de *aid*, empleando la vista recursiva de SQL que aparece en la Figura 5.15.

Todas las vistas recursivas deben definirse como la unión de dos subconsultas: una **consulta base** que no es recursiva y una **consulta recursiva** que utiliza la vista recursiva. En el ejemplo de la Figura 5.15 la consulta base es la selección sobre *prerreq*, mientras que la consulta recursiva calcula la reunión de *prerreq* y *rec_prerreq*.

El significado de las vistas recursivas se comprende mejor de la siguiente manera: en primer lugar hay que calcular la consulta base y añadir todas las tuplas resultantes a la vista definida recursivamente *rec_prerreq;* a continuación hay que calcular la consulta recursiva empleando el contenido actual de la vista y añadir todas las tuplas resultantes a la vista. Se debe seguir repitiendo este paso hasta que no se añada ninguna tupla nueva a la vista. La instancia resultante de la vista se denomina **punto fijo** de la definición recursiva de la vista. (El término «fijo» hace referencia al hecho de que ya no se producen más modificaciones). La vista, por tanto, se define para que contenga exactamente las tuplas de la instancia del punto fijo.

Aplicando la lógica anterior a nuestro ejemplo, se encuentran primero todos los prerrequisitos directos de cada asignatura ejecutando la consulta base. La consulta recursiva añade un nivel de asignaturas en cada iteración, hasta alcanzar la profundidad máxima de la relación asignatura-prerrequisito. En ese punto ya no se añaden nuevas tuplas a la vista y la iteración ha alcanzado un punto fijo.

Para encontrar los prerrequisitos de una determinada asignatura, como la CS-347, se puede modificar la consulta del nivel externo añadiendo una cláusula «**where** *rec_prerreq.asignatura_id* = ´CS-347´». Una forma de evaluar la consulta con la selección es calcular el contenido completo de *rec_prerreq* usando la técnica iterativa, y después seleccionar de este resultado solo las tuplas cuyo *asignatura_id* sea CS-347. Sin embargo, esto necesitaría calcular los pares (asignatura, prerrequisito) de todas las asignaturas, todos ellos irrelevantes excepto para la asignatura CS-347. De hecho no se exige que el sistema de bases de datos utilice esta técnica iterativa para calcular el resultado completo de la consulta recursiva y después realizar la selección. Se puede obtener el mismo resultado empleando otras técnicas que pueden ser más eficientes, como las que se utilizan en la función *buscarTodosReq* que se vio anteriormente. Consulte las notas bibliográficas para obtener más información sobre este tema.

Hay algunas restricciones para la consulta recursiva en una vista recursiva, concretamente, la consulta debe ser **monótona,** es decir, su resultado sobre la instancia de una relación vista V_1 debe ser un superconjunto de su resultado sobre la instancia de una relación vista V_2 si V_1 es un superconjunto de V_2. De manera intuitiva, si se añaden más tuplas a la relación vista, la consulta recursiva debe devolver, como mínimo, el mismo conjunto de tuplas que antes, y posiblemente devuelva tuplas adicionales.

En concreto, las consultas recursivas no deben emplear ninguno de los constructores siguientes, ya que harían que la consulta no fuera monótona:

- Agregación sobre la vista recursiva.
- **not exists** sobre una subconsulta que utiliza la vista recursiva.
- Diferencia de conjuntos **(except)** cuyo lado derecho utilice la vista recursiva.

Por ejemplo, si la consulta recursiva fuera de la forma $r - v$, donde v es la vista recursiva, si se añade una tupla a v, el resultado de la consulta puede hacerse más pequeño; la consulta, por tanto, no es monótona.

El significado de las vistas recursivas se puede definir mediante el procedimiento recursivo mientras la consulta recursiva sea monótona; si la consulta recursiva no es monótona, el significado de la consulta resulta difícil de definir. SQL, por tanto, exige que las consultas sean monótonas. Las consultas recursivas se estudian con más detalle en el contexto del lenguaje de consultas Datalog, en la Sección C.3.6 (Apéndice C, en red).

SQL también permite la creación de vistas permanentes definidas de manera recursiva mediante el uso de **create recursive view,** en lugar de **with recursive.** Algunas implementaciones soportan consultas recursivas que emplean una sintaxis diferente; consulte el manual del sistema correspondiente para conocer más detalles.

5.5. Características de agregación avanzadas **

Las posibilidades de agregación en SQL que se han visto anteriormente son muy potentes y permiten manejar la mayoría de las tareas con facilidad. Sin embargo, existen tareas que resultan difíciles de implementar eficientemente con las características de agregación básicas. En esta sección se estudian las características que se añadieron a SQL para manejar estas tareas.

```
with recursive a_prerreq(asignatura_id, prerreq_id) as (
    select asignatura_id, prerreq_id
    from prerreq
  union
    select prerreq.prerreq_id, a_prerreq.asignatura_id
    from prerreq, a_prerreq
    where prerreq.asignatura_id = a_prerreq.prereq_id
  )
select *
from a_prerreq;
```

Figura 5.15. Consulta recursiva en SQL.

5.5.1. Clasificación

Una operación habitual es encontrar la posición que ocupa un valor dentro de un conjunto grande de datos. Por ejemplo, se puede querer asignar un puesto en la clase a los estudiantes dependiendo de su nota media (NM), dando el puesto 1 al estudiante con mayor nota media, el puesto 2 al de siguiente media, etc. Un tipo de consulta relacionada consiste en encontrar el percentil en el que se encuentra un determinado valor en un (multi)conjunto, por ejemplo, el tercio inferior, el tercio medio o el tercio superior. Aunque estas consultas se pueden expresar utilizando las construcciones de SQL que se han visto hasta este momento, resultan difíciles de expresar e ineficientes de evaluar. Los programadores pueden decidir escribir la consulta parcialmente en SQL y parcialmente en un lenguaje de programación. Se estudiará el soporte de SQL para expresar directamente este tipo de consultas.

En el ejemplo de la universidad, la relación *matricula* incluye la nota que ha obtenido cada estudiante en cada asignatura. Para mostrar la clasificación, suponga que se tiene una vista *estudiante_nota* (*ID*, *NM*) con la nota media de cada estudiante.[8]

La clasificación se realiza con **order by.** La siguiente consulta devuelve la posición de cada estudiante:

select *ID*, **rank**() **over** (**order by** (*NM*) **desc**) **as** *e_pos*
from *estudiante_nota*;

Fíjese que el orden de las tuplas en el resultado no está definido, por lo que pueden aparecer sin ordenar por posición. Se necesita una cláusula adicional de **order by** para que queden ordenados, como se muestra a continuación:

select *ID*, **rank** () **over** (**order by** (*NM*) **desc**) **as** *e_pos*
from *estudiante_nota*
order by *e_pos*;

Un elemento básico en las clasificaciones es cómo tratar el caso de varias tuplas que son iguales por el atributo o atributos de ordenación. En el ejemplo, significa decidir qué hacer con dos estudiantes con la misma nota media. La función **rank** da la misma posición a todas las tuplas que son iguales por el atributo de **order by.** Por ejemplo, si hay dos estudiantes que tienen la mayor nota media, ambos tendrán la posición 1. La siguiente posición será la 3, no la 2, por lo que si hay tres estudiantes con la misma nota media siguiente, los tres tendrán la posición 3, y el siguiente estudiante tendrá la posición 5, y así sucesivamente. También existe una función **dense_rank** que no crea huecos en el orden. Siguiendo el ejemplo anterior, las tuplas con el segundo mayor valor tendrán la posición 2, y las tuplas con el tercer mayor valor la posición 3, y así sucesivamente.

Se puede expresar la consulta anterior con las funciones de agregación básicas de SQL, mediante la siguiente consulta:

select *ID*, (1+ (**select count**(*)
from *estudiante_nota B*
where *B.GPA* > *A.GPA*)) **as** *e_pos*
from *estudiante_nota A*
order by *e_pos*;

Debería quedar claro que la posición de un estudiante es simplemente 1 más el número de estudiantes con una mayor nota media, que es exactamente lo que especifica la consulta anterior. Sin embargo, este cálculo de la posición de cada estudiante es lineal en tiempo con respecto al tamaño de la relación, lo que genera que sea cuadrático en tiempo con respecto al tamaño. Para relaciones grandes, la consulta anterior tardaría mucho tiempo en ejecutarse. Por otra parte, la implementación del sistema de la cláusula **rank** puede ordenar la relación y calcular la clasificación en mucho menos tiempo.

La clasificación se puede realizar mediante particiones de los datos. Por ejemplo, suponga que se desea clasificar a los estudiantes por departamento, en lugar de para toda la universidad. Suponga que se define una vista como *estudiante_nota* pero incluyendo el nombre del departamento: *dept_nota*(*ID*, *nombre_dept*, *NM*). La siguiente consulta calcula la clasificación de los estudiantes de cada sección:

select *ID*, *nombre_dept*,
rank () **over** (**partition by** *nombre_dept* **order by** *NM* **desc**)
as *dept_pos*
from *dept_nota*
order by *nombre_dept*, *dept_pos*;

La cláusula **order by** externa ordena los resultados de las tuplas por el nombre de departamento, y dentro de cada departamento, por la posición.

Se pueden utilizar varias expresiones **rank** en una única sentencia **select;** por tanto, se puede obtener la clasificación global y la clasificación dentro del departamento utilizando dos expresiones **rank** en la misma cláusula **select.** Cuando se realiza una clasificación (posiblemente con particionado) junto con una cláusula **order by,** se aplica primero la cláusula **group by** y el particionado y la clasificación se realizan sobre el resultado de la cláusula **group by.** Por tanto, se pueden utilizar los valores agregados para realizar la clasificación. Se podría haber escrito la clasificación sobre la vista *estudiante_nota* sin utilizar la vista, usando una simple cláusula **select.** Se dejan los detalles como ejercicio para el lector.

Las funciones de clasificación se pueden utilizar para encontrar las primeras n tuplas incluyendo una consulta de clasificación dentro de una consulta externa; se dejan los detalles como ejercicio. Fíjese que calcular las últimas n tuplas es simplemente igual que las primeras n tuplas con una ordenación al revés. Algunos sistemas de bases de datos proporcionan extensiones no estándar de SQL para especificar directamente que solo se desean los primeros n resultados; estas extensiones no requieren utilizar la función de clasificación y simplifican la tarea al optimizador. Por ejemplo, algunas bases de datos permiten añadir una cláusula **limit** n al final de una sentencia de SQL para especificar que solo se deberían generar las primeras n tuplas; esta cláusula se utiliza en conjunto con la cláusula **order by** para obtener las primeras n tuplas, como se muestra en la siguiente consulta, que obtiene los *ID* y *NM* de los primeros diez estudiantes por orden de *NM*.

select *ID*, *NM*)
from *estudiante_nota*
order by *NM*
limit 10;

Sin embargo, la cláusula **limit** no permite la partición, por lo que no se pueden obtener los primeros n dentro de cada partición sin realizar una clasificación; más aún si existe más de un estudiante con la misma nota media, ya que es posible que uno aparezca en los diez primeros y el otro no.

Se pueden utilizar otras funciones en el lugar de **rank.** Por ejemplo, **percent_rank** de una tupla indica la posición de la tupla como una fracción. Si existen n tuplas de la partición[9] y la posición de la tupla es r, entonces su posición porcentual se define como $(r - 1)/(n - 1)$ (y como *null* si solo existe una tupla en la partición). La función **cume_dist,** abreviatura de distribución acumulativa, para una tupla se define como p/n, donde p es el número de tuplas de la partición con valores de ordenación anteriores o iguales al valor de

8 La sentencia SQL para crear la vista *estudiante_nota* es un tanto compleja, ya que hay que convertir las notas en letras de la relación *matricula* a números y ponderar la nota de cada asignatura según el número de créditos de cada una. La definición de esta vista es el objetivo del Ejercicio 4.5.

9 Si no se utiliza ninguna partición explícita el conjunto se trata como una única partición.

ordenación de la tupla y *n* es el número de tuplas en la partición. La función **row_number** ordena las filas y da a cada fila un número único que se corresponde con su posición en el orden de salida; las filas que son distintas y tienen el mismo valor de ordenación obtienen un número diferente de fila, de forma no determinista.

Por último, dada una constante *n*, la función de clasificación **ntile**(*n*) dadas las tuplas de cada partición el es orden especificado, las divide en *n* grupos con el mismo número de tuplas.[10] Para cada una de las tuplas, **ntile**(*n*) proporciona el número del grupo en que ha acabado, empezando a numerar los grupos con el número 1. Esta función es particularmente útil para construir histogramas según los percentiles. Se puede calcular el cuartil en que se encuentra cada estudiante según su nota media con la siguiente consulta:

select *ID*, **ntile**(4) **over** (**order by** (*NM* **desc**)) **as** *quartile*
from *estudiante_nota*;

Si existen valores nulos, la definición de una clasificación se puede complicar, pues no queda claro qué se debería considerar primero en el orden de clasificación. SQL permite que el usuario especifique dónde se deberían ubicar los nulos utilizando **nulls first** o **nulls last,** por ejemplo:

select *ID*, **rank** () **over** (**order by** *NM* **desc nulls last**) **as** *e_pos*
from *estudiante_nota*;

5.5.2. Ventanas

Las consultas de ventana calculan una función agregada sobre un intervalo de tuplas. Es útil, por ejemplo, para calcular un agregado sobre un intervalo fijo de tiempo; el intervalo de tiempo se denomina una *ventana*. Las ventanas se pueden solapar, en cuyo caso una tupla puede contribuir a más de una ventana. Este caso es distinto del anterior, en el que una tupla solo podía contribuir a una de las particiones.

Un ejemplo de uso de las ventanas es el análisis de tendencia. Considere el ejemplo de ventas anterior. Las ventas pueden cambiar de forma importante de un día a otro por distintos factores, como el tiempo (por ejemplo una tormenta de nieve, una inundación, un huracán o un terremoto podrían reducir las ventas durante un periodo de tiempo). Sin embvargo, teniendo en cuenta un periodo de tiempo suficientemente amplio, las fluctuaciones serán menores (continuando con el ejemplo, las ventas se «rehacen» con los cambios del tiempo. El análisis de tendencias del mercado de valores es otro ejemplo del uso del concepto de ventana. En los sitios de negocios e inversión existen distintas «medias móviles».

Resulta relativamente sencillo escribir una consulta de SQL utilizando las características que ya se han estudiado para calcular un agregado sobre una ventana, por ejemplo de ventas, durante periodos fijos de tres días. Sin embargo, si se desea realizar este calculo para *todos* los periodos de tres días, la consulta resulta compleja.

SQL proporciona una característica de ventana para estas consultas. Suponga que se dispone de una ventana *total_créditos*(*año*, *núm_créditos*) que indica el número total de créditos en que se ha matriculado cada estudiante cada año.[11] Fíjese que esta relación puede contener como mucho una tupla por año. Considere la siguiente consulta:

select *año*, **avg**(*núm_créditos*)
  over (**order by** *año* **rows** 3 **preceding**)
  as *med_total_créditos*
from *total_créditos*;

Esta consulta calcula el promedio de las tres tuplas *precedentes* en el orden especificado. Por tanto, para 2009, si las tuplas de los años 2008 y 2007 existen en la relación *total_créditos*, con cada año representado por una sola tupla, el resultado de la definición de ventana es la media de los valores de los años 2007, 2008 y 2009.

Los promedios de cada año se calcularían de forma similar. Para el primer año de la relación *total_créditos*, la media sería solo la de ese año, mientras que para el siguiente año la media se calcularía sobre los dos años.

Fíjese que si la relación *total_créditos* tiene más de una tupla para un año dado, existen varias posibilidades de ordenación de las tuplas. En este caso, la definición de tuplas precedentes depende de la implementación de la ordenación y no está definida de forma única.

Suponga que en lugar de ir hacia atrás un número fijo de tuplas, se desea que la ventana consista en todos los años anteriores. Esto significa que el número de años anteriores considerados no es fijo.

Para obtener el promedio de créditos totales en todos los años anteriores se escribe:

select *año*, **avg**(*núm_créditos*)
  over (**order by** *año* **rows unbounded preceding**)
  as *med_total_créditos*
from *total_créditos*;

Se puede utilizar la palabra clave **following** en lugar de **preceding.** Si se hace en el ejemplo, el valor *año* indica el principio de la ventana en lugar del final. De forma similar, se puede especificar el principio de la ventana antes de la tupla actual y el final tras ella:

select *año*, **avg**(*núm_créditos*)
  over (**order by** *año* **rows between** 3 **preceding**
    and 2 **following**)
  as *med_total_créditos*
from *total_créditos*;

En lugar de especificar una cuenta de tuplas, se puede especificar un intervalo de acuerdo con el atributo **order by.** Para especificar un intervalo de cuatro años hacia atrás, incluyendo el año actual, se escribe:

select *año*, **avg**(*núm_créditos*)
  over (**order by** *año* **range between** *año* -4 **and** *año*)
  as *med_total_créditos*
from *total_créditos*;

Fíjese en el uso de la palabra clave **range** en el ejemplo anterior. Para el año 2010, se incluirían los datos de los años 2006 a 2010, inclusive, independientemente de cuántas tuplas existiesen en dicho intervalo.

En el ejemplo, todas las tuplas pertenecen a toda la universidad. Suponga que hubiese datos de créditos para cada departamento en una vista *total_créditos_dept*(*nombre_dept*, *año*, *núm_créditos*) con el número total de créditos que el estudiante ha cursado en un determinado departamento y en un determinado año. (De nuevo, se deja cómo escribir la definición de esta vista como ejercicio).

Se pueden escribir consultas de ventana que traten cada departamento de forma separada dividiendo por *nombre_dept*:

select *nombre_dept*, *año*, **avg**(*núm_créditos*)
  over (**partition by** *nombre_dept*
    order by *year* **rows between** 3 **preceding and current row**)
  as *med_total_créditos*
from *total_créditos_dept*;

10 Si el número total de tuplas en la partición no es divisible por *n*, entonces el número de tuplas de cada grupo puede diferir como mucho en 1. Las tuplas con el mismo valor del atributo de ordenación se pueden asignar a diferentes grupos de forma no determinista para que el número de tuplas de cada grupo sea el mismo.

11 Se deja la definición de esta vista del ejemplo de la universidad como ejercicio.

5.6. OLAP**

Un sistema de procesamiento analítico en línea (*online analytical processing*: OLAP) es un sistema interactivo que permite a un analista ver diferentes resúmenes de datos multidimensionales. La referencia *en línea* indica que el analista ha de ser capaz de solicitar nuevos resúmenes y obtener respuestas en línea en pocos segundos, sin que se vea obligado a esperar mucho tiempo para obtener los resultados de su consulta.

Existen muchos productos OLAP disponibles, incluyendo algunos que se incluyen con los productos de bases de datos como SQL Server de Microsoft y Oracle, y otros que son herramientas independientes. Las versiones iniciales de OLAP asumían que los datos se encontraban residentes en la memoria. El análisis de datos sobre cantidades de datos pequeñas se puede, de hecho, realizar con herramientas de hoja de cálculo como Excel. Sin embargo, sobre grandes cantidades de datos OLAP necesita que estos se encuentren residentes en una base de datos y requiere el soporte de una base de datos para un preprocesado eficiente de los datos, así como para el procesamiento de consultas en línea. En esta sección se estudian las extensiones de SQL para estas tareas.

5.6.1. Procesamiento analítico en línea

Considere una aplicación en la que una tienda quiere descubrir qué tipos de ropa son los más populares. Supongamos que la ropa se caracteriza por su nombre de elemento, color y tamaño, y se dispone de una relación *ventas* con el esquema:

ventas(*nombre_elem*, *color*, *talla*, *cantidad*)

Suponga que el *nombre_elem* tiene los valores (falda, vestido, camisa, pantalón), *color* tiene los valores (oscuro, pastel, blanco), *talla* tiene los valores (pequeño, mediano, grande) y *cantidad* es un valor entero que representa el número total de elementos vendidos de un tipo {*nombre_elem*, *color*, *talla*}. Un ejemplar de la relación *ventas* se muestra en la Figura 5.16.

El análisis estadístico suele requerir el agrupamiento por varios atributos. Dada una relación que es utiliza para el análisis de datos, se pueden identificar algunos de sus atributos como atributos de **medida,** ya que miden algún valor, y se puede realizar la agregación por ellos. Por ejemplo, el atributo *cantidad* de la relación *ventas* es un atributo de medida, ya que mide el número total de unidades vendidas. Algunos (o todos) de los atributos restantes de la relación se identifican como **atributos de dimensión,** ya que definen las dimensiones sobre las que observar los atributos de medida y los resúmenes de los atributos de medida. En la relación *ventas*, *nombre_elem*, *color* y *talla* son atributos de dimensión. (Una versión más realista de la relación *ventas* tendría dimensiones adicionales como la ubicación de la venta, y medidas adicionales como el valor monetario de la venta).

Los datos que se pueden modelar como atributos de dimensión y atributos de medida se denominan **datos multidimensionales.**

Para analizar los datos multidimensionales, un gestor puede querer ver los datos ordenados como se muestra en la tabla de la Figura 5.17. La tabla muestra la cantidad total de las distintas combinaciones de *nombre_elem* y *color*. El valor de *talla* se especifica que sea **todos,** indicando que los valores que se muestran son un resumen de todos los valores de *talla* (es decir, se desea agrupar los elementos «pequeño», «mediano» y «grande» en un solo grupo).

La tabla de la Figura 5.17 es un ejemplo de una **tabla-cruzada** (o *cross-tab*), también denominada **tabla dinámica** (*pivot-table*). En general, una tabla cruzada es una tabla derivada de una relación (digamos R), en la que los valores de un atributo de la relación (digamos A) forman la cabecera de las filas y los valores de otro atributo de la relación R (digamos B) forman la cabecera de las columnas. Por ejemplo, en la Figura 5.17 el atributo *nombre_elem* se corresponde con A (con los valores «oscuro», «pastel» y «blanco») y el atributo *color* se corresponde con B (con los atributos «falda», «vestido», «camisa» y «pantalón»).

Cada celda de la tabla dinámica se puede identificar con (a_i, b_j), donde a_i es un valor de A, y b_j es un valor de B. Los valores de las distintas celdas de la tabla dinámica se derivan de la relación R de la siguiente forma: si existe como mucho una tupla en R con cualquier valor de (a_i, b_j) el valor de la celda se deriva de esa única tupla (si existe); por ejemplo, sería el valor de uno o más del resto de los atributos de la tupla. Si hubiese varias tuplas con el valor (a_i, b_j) el valor de la celda se debe derivar por agregación de las tuplas con ese valor. En el ejemplo, la agregación utilizada es la suma de los valores del atributo *cantidad* para todos los valores de *talla*, como se indica por *talla*:**all** en la tabla cruzada de la Figura 5.17. Por tanto, el valor de la celda (falda, pastel) es 35, ya que existen tres tuplas en la tabla *ventas* que coinciden con ese criterio, con valores 11, 9 y 15.

nombre_elem	*color*	*talla*	*cantidad*
falda	oscuro	pequeño	2
falda	oscuro	mediano	5
falda	oscuro	grande	1
falda	pastel	pequeño	11
falda	pastel	mediano	9
falda	pastel	grande	15
falda	blanco	pequeño	2
falda	blanco	mediano	5
falda	blanco	grande	3
vestido	oscuro	pequeño	2
vestido	oscuro	mediano	6
vestido	oscuro	grande	12
vestido	pastel	pequeño	4
vestido	pastel	mediano	3
vestido	pastel	grande	3
vestido	blanco	pequeño	2
vestido	blanco	mediano	3
vestido	blanco	grande	0
camisa	oscuro	pequeño	2
camisa	oscuro	mediano	6
camisa	oscuro	grande	6
camisa	pastel	pequeño	4
camisa	pastel	mediano	1
camisa	pastel	grande	2
camisa	blanco	pequeño	17
camisa	blanco	mediano	1
camisa	blanco	grande	10
pantalón	oscuro	pequeño	14
pantalón	oscuro	mediano	6
pantalón	oscuro	grande	0
pantalón	pastel	pequeño	1
pantalón	pastel	mediano	0
pantalón	pastel	grande	1
pantalón	blanco	pequeño	3
pantalón	blanco	mediano	0
pantalón	blanco	grande	2

Figura 5.16. Ejemplo de la relación *ventas*.

En el ejemplo, la tabla cruzada también tiene una columna y una fila adicionales con los totales de las celdas de cada fila y columna. La mayor parte de las tablas cruzadas tienen esas filas y columnas de resumen.

La generalización de una tabla cruzada, que es bidimensional, a n dimensiones se puede visualizar como cubos n-dimensionales, denominados **cubos de datos.** La Figura 5.18 muestra un cubo de datos para la relación *ventas*. El cubo de datos tiene tres dimensiones, *nombre_elem*, *color* y *talla*, y el atributo de medida es *cantidad*. Cada celda se identifica por los valores de estas tres dimensiones. Cada celda del cubo de datos contiene un valor, igual que en la tabla cruzada. En la Figura 5.18 el valor de la celda se muestra en una de las caras de la celda; las otras caras de la celda, si son visibles, se muestran en blanco. Todas las celdas contienen valores, aunque no sean visibles. El valor de una dimensión puede ser **all,** en cuyo caso la celda contiene un resumen de todos los valores de esa dimensión, como en el caso de las tablas cruzadas.

El número de maneras diferentes en que las tuplas pueden agruparse para su agregación puede ser grande. En el ejemplo de la Figura 5.18 existen tres colores, cuatro elementos y tres tallas, lo que genera un cubo de tamaño $3 \times 4 \times 3 = 36$. Incluyendo los valores resumen se obtiene un cubo de tamaño $4 \times 5 \times 4$, que es de tamaño 80. De hecho, para una tabla de n dimensiones, la agregación se puede realizar con agrupamientos de cada uno de los 2_n subconjuntos de las n dimensiones.[12]

Con los sistemas OLAP los analistas de datos pueden ver diferentes tablas cruzadas para los mismos datos seleccionando de manera interactiva los atributos de las tablas cruzadas. Cada tabla cruzada es una vista bidimensional del cubo de datos multidimensional. Por ejemplo, el analista puede seleccionar una tabla cruzada para *nombre_elem* y *talla* o una tabla cruzada para *color* y *talla*. La operación de modificación de las dimensiones utilizadas en las tabulaciones cruzadas se denomina **pivotaje.**

Los sistemas OLAP permiten al analista ver una tabla cruzada de *nombre_elem* y *color* para un valor fijo de *talla*, por ejemplo, grande, en vez de la suma de todas las tallas. Esta operación se denomina **división** o **corte,** pues se contempla como visualización de un corte practicado en el cubo de datos. A veces, a esta operación se le llama **creación de datos,** particularmente cuando se fijan los valores para dimensiones múltiples.

Cuando se utilizan tablas cruzadas para ver cubos multidimensionales los valores de los atributos de dimensión que no forman parte de la tabla cruzada se muestran por encima de ella. El valor de estos atributos puede ser **all,** como en la Figura 5.17, lo que indica que los datos de la tabulación cruzada son un resumen de todos los valores del atributo. El corte y la creación de datos consisten simplemente en la selección de valores concretos de estos atributos, que se muestran luego en la parte superior de la tabla cruzada.

Los sistemas OLAP permiten a los usuarios examinar los datos con el nivel de granularidad deseado. La operación de pasar de datos con una granularidad más fina a una granularidad más gruesa (mediante la agregación) se denomina **abstracción.** En este ejemplo, a partir del cubo de datos para la tabla *ventas* se obtiene la tabla cruzada de ejemplo abstrayendo el atributo *talla*. La operación inversa, la de pasar de datos con una granularidad más gruesa a una más fina, se denomina **concreción.** Claramente, los datos con granularidad más fina no se pueden generar a partir de datos con una granularidad más gruesa; deben generarse a partir de los datos originales o de datos resumidos de granularidad aún más fina.

Puede que un analista desee examinar una dimensión con niveles diferentes de detalle. Por ejemplo, los atributos de tipo **FechaHora** contienen una fecha y una hora del día. El empleo de horas con precisión de segundos (o menos) puede que no sea significativo: los analistas que estén interesados en la hora aproximada del día puede que solo miren el valor de hora. Los analistas que estén interesados en las ventas de cada día de la semana puede que apliquen la fecha al día de la semana y solo se fijen en eso. Puede que otro analista esté interesado en agregados mensuales, trimestrales o de años enteros.

Los diferentes niveles de detalle de los atributos pueden organizarse en una **jerarquía.** La Figura 5.19a muestra una jerarquía para el atributo **FechaHora.** La Figura 5.19b, que puede ser otro ejemplo, muestra una jerarquía para la ubicación, con la ciudad en la parte inferior de la jerarquía, la provincia por encima, el país en el nivel siguiente y la región en el nivel superior. En el ejemplo anterior la ropa puede agruparse por categorías (por ejemplo, ropa de hombre o de mujer); la *categoría* estaría por encima de *nombre_elem* en la jerarquía de la ropa. En el nivel de los valores reales las faldas y los vestidos caerían dentro de la categoría de ropa de mujer, y los pantalones y las camisas en la de ropa de hombre.

talla **all**

nombre_elem \ *color*	oscuro	pastel	blanco	total
falda	8	35	10	53
vestido	20	10	5	35
camisa	14	7	28	49
pantalón	20	2	5	27
total	62	54	48	164

Figura 5.17. Tabla cruzada de *ventas* por *nombre_elem* y *color*.

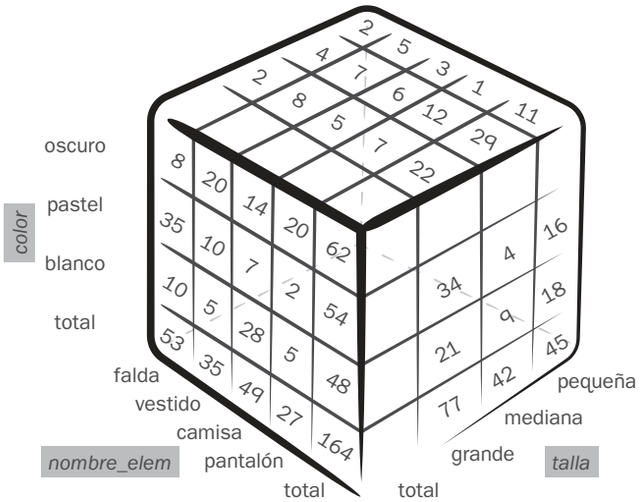


Figura 5.18. Cubo de datos tridimensional.

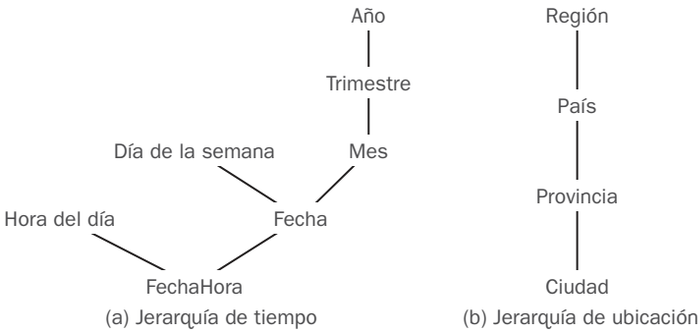


Figura 5.19. Jerarquías de las dimensiones.

12 La agrupación sobre el conjunto de las n dimensiones solo resulta útil si la tabla puede tener duplicados.

IMPLEMENTACIÓN DE OLAP

Los primeros sistemas de OLAP utilizaban arrays de memoria multidimensionales para almacenar los cubos de datos y se denominaban sistemas OLAP **multidimensionales (Multidimensional OLAP, MOLAP).** Posteriormente, los servicios OLAP se integraron en los sistemas relacionales y los datos se almacenaron en las bases de datos relacionales. Estos sistemas se denominan sistemas **OLAP relacionales (Relational OLAP, ROLAP).** Los sistemas híbridos, que almacenan algunos resúmenes en la memoria y los datos básicos y otros resúmenes en bases de datos relacionales, se denominan sistemas **OLAP híbridos (Hybrid OLAP, HOLAP).**

Muchos sistemas OLAP se implementan como sistemas cliente-servidor. El servidor contiene la base de datos relacional así como los cubos de datos MOLAP. Los sistemas clientes obtienen vistas de los datos comunicándose con el servidor.

Una manera ingenua de calcular todo el cubo de datos (todas las agrupaciones) de una relación es utilizar cualquier algoritmo estándar para calcular las operaciones de agregación, agrupación a agrupación. El algoritmo ingenuo necesitaría un gran número de exploraciones de la relación. Una optimización sencilla consiste en calcular la agregación para, por ejemplo, (*nombre_elem*, *color*) a partir del agregado (*nombre_elem*, *color*, *talla*), en lugar de hacerlo a partir de la relación original.

Para las funciones de agregación estándar de SQL se pueden calcular agregados con agrupaciones sobre un conjunto de atributos A a partir de un agregado con agrupación sobre un conjunto de atributos B si $A \subseteq B$; se puede hacer como ejercicio (consulte el Ejercicio 5.24), pero hay que tener en cuenta que al calcular **avg** también es necesario el valor **count** (para algunas funciones de agregación no estándar como la mediana, los agregados no se pueden calcular de la manera indicada; la optimización aquí descrita no es aplicable a estas funciones de agregación *no descomponibles*). La cantidad de datos que se leen disminuye de manera significativa al calcular los agregados a partir de otros agregados, en lugar de hacerlo a partir de la relación original. Se pueden conseguir otras mejoras; por ejemplo, se pueden calcular varias agrupaciones con una sola lectura de los datos.

Las primeras implementaciones de OLAP calculaban previamente los cubos de datos completos, es decir, las agrupaciones sobre todos los subconjuntos de los atributos de dimensión, y los almacenaban. El cálculo previo permite que las consultas OLAP se respondan en pocos segundos, incluso para conjuntos de datos que pueden contener millones de tuplas que suponen gigabytes de datos. No obstante, hay 2^n agrupaciones con n atributos de dimensión; las jerarquías sobre los atributos aumentan aún más el número. En consecuencia, todo el cubo de datos suele ser mayor que la relación original que lo generó y, en muchos casos, no resulta posible almacenarlo entero.

En lugar de calcular previamente todas las agrupaciones posibles y almacenarlas, resulta razonable calcular previamente algunas de las agrupaciones y almacenarlas, y calcular el resto según se soliciten. En lugar de calcular las consultas a partir de la relación original, lo que puede tardar mucho tiempo, se pueden calcular a partir de otras consultas calculadas previamente. Por ejemplo, suponga que una consulta necesita resúmenes según (*nombre_elem*, *color*), que no se ha calculado con anterioridad. El resultado de la consulta puede calcularse a partir de resúmenes según (*nombre_elem*, *color*, *talla*), si ya se ha calculado. Consulte las notas bibliográficas para más referencias sobre el modo de seleccionar un buen conjunto de agrupaciones para su cálculo previo, dados los límites de almacenamiento disponible para los resultados calculados previamente.

Puede que un analista esté interesado en consultar las ventas de ropa divididas entre ropa de hombre y ropa de mujer y que no esté interesado en sus valores individuales. Tras ver los agregados en el nivel de ropa de hombre y ropa de mujer puede que el analista *concrete la jerarquía* (*drill down*) para ver los valores individuales. Un analista que examine el nivel detallado puede *abstraer la jerarquía* y examinar agregados de niveles más gruesos. Ambos niveles pueden mostrarse en la misma tabla cruzada, como en la Figura 5.20.

talla **all**

categoría	*nombre_elem*	*color* oscuro	pastel	blanco	total
ropa de mujer	falda	8	35	10	53
	vestido	20	10	5	35
	subtotal	28	45	15	88
ropa de hombre	camisa	14	7	28	49
	pantalón	20	2	5	27
	subtotal	34	9	33	76
total		62	54	48	164

Figura 5.20. Tabla cruzada de *ventas* con la jerarquía sobre *nombre_elem*.

5.6.2. Tablas cruzadas y tablas relacionales

Las tablas cruzadas son diferentes de las tablas relacionales que suelen guardarse en las bases de datos, ya que el número de columnas de las tablas cruzadas depende de los datos. Una modificación en los valores de los datos puede dar lugar a que se añadan más columnas, lo que no resulta deseable para el almacenamiento de los datos. No obstante, la vista de tabla cruzada es adecuada para mostrársela a los usuarios. La representación de las tablas cruzadas sin valores resumen en un formulario relacional con un número fijo de columnas es directa. La tabla cruzada con columnas o filas resumen puede representarse introduciendo el valor especial **all** para representar los subtotales, como en la Figura 5.21. La norma SQL:1999 utiliza realmente el valor **null** (nulo) en lugar de **all,** pero, para evitar confusiones con los valores nulos habituales, en el libro se seguirá utilizando **all.**

nombre_elem	*color*	*talla*	*cantidad*
falda	oscuro	**all**	8
falda	pastel	**all**	35
falda	blanco	**all**	10
falda	**all**	**all**	53
vestido	oscuro	**all**	20
vestido	pastel	**all**	10
vestido	blanco	**all**	5
vestido	**all**	**all**	35
camisa	oscuro	**all**	14
camisa	pastel	**all**	7
camisa	blanco	**all**	28
camisa	**all**	**all**	49
pantalón	oscuro	**all**	20
pantalón	pastel	**all**	2
pantalón	blanco	**all**	5
pantalón	**all**	**all**	27
all	oscuro	**all**	62
all	pastel	**all**	54
all	blanco	**all**	48
all	**all**	**all**	164

Figura 5.21. Representación relacional de los datos de la Figura 5.17.

Considere las tuplas (falda, **all, all,** 53) y (vestido, **all, all,** 35). Se han obtenido estas tuplas eliminando las tuplas individuales con diferentes valores de *color* y *talla*, y sustituyendo el valor de *cantidad* por un agregado, la suma de las cantidades. El valor **all** puede considerarse representante del conjunto de los valores de un atributo. Las tuplas con el valor **all** para las dimensiones *color* y *talla* pueden obtenerse mediante una agregación de la relación *ventas* con un **group by** sobre la columna *nombre_elem*.

De forma parecida, se puede utilizar **group by** sobre *color* y *talla* para conseguir las tuplas con el valor **all** para *nombre_elem*, y se puede utilizar **group by** sin atributos (que en SQL puede omitirse simplemente) para obtener la tupla con el valor **all** para *nombre_elem*, *color* y *talla*.

Las jerarquías también se pueden representar mediante relaciones. Por ejemplo, el hecho de que faldas y vestidos estén en la categoría ropa de mujer, y pantalón y camisas en la categoría ropa de hombre se puede representar mediante una relación *categoría_elemento*(*nombre_elem*, *categoría*). Esta relación se puede hacer una reunión con la relación *ventas* para obtener una relación que incluya la categoría de cada elemento. La agregación sobre esta relación reunida permite obtener una tabla cruzada con la jerarquía.

Otro ejemplo, una jerarquía sobre ciudad se puede representar mediante una sola relación *jerarquía_ciudad*(*ID*, *ciudad*, *provincia*, *país*, *región*), o mediante varias relaciones, cada una asociando valores de un nivel de la jerarquía con valores del nivel siguiente. Se supone que las ciudades tienen identificadores únicos, que se guardan con el atributo *ID*, para evitar confusiones entre dos ciudades con el mismo nombre; por ejemplo, Springfield de Missouri con Springfield de Illinois.

5.6.3. OLAP en SQL

Varias implementaciones de SQL, como SQL Server de Microsoft y Oracle, soportan una cláusula **pívot** en SQL, que permite la creación de tablas cruzadas. Dada la relación *ventas* de la Figura 5.16, la consulta:

```
select *
from ventas
pivot (
    sum(cantidad)
    for color in (´oscuro´,´pastel´,´blanco´)
)
order by nombre_elem;
```

devuelve la tabla cruzada de la Figura 5.22.

Fíjese que la cláusula **for** dentro de la cláusula **pívot** especifica qué valores del atributo *color* deberían aparecer como nombres de atributos en el resultado dinámico. El propio atributo *color* se elimina del resultado, aunque el resto de los atributos se mantienen, excepto que los valores para el nuevo atributo creado se especifica que provengan del atributo *cantidad*. En el caso de que más de una tupla contribuya a los valores de una celda, la operación de agregación dentro de la cláusula **pívot** especifica cómo se deberían combinar los valores. En el ejemplo anterior, se suman los valores de *cantidad*.

Observe que la propia cláusula **pívot** no calcula los subtotales que había en la tabla dinámica de la Figura 5.17. Sin embargo, primero se puede generar la representación relacional que se muestra en la Figura 5.21, como se ha mostrado, y después se aplica la cláusula **pívot** sobre esta representación para obtener un resultado equivalente. En este caso, el valor **all** también se debe listar en la cláusula **for,** y se necesita modificar la cláusula **order by** para ordenar **all** al final.

Los datos de un cubo de datos no se pueden generar mediante una única consulta de SQL, utilizando la construcción **group by,** ya que los agregados se calculan para varios agrupamientos diferentes sobre los atributos dimensiones. Por esta razón, SQL incluye funciones para formar los agrupamientos necesarios para OLAP. Este tema se tratará más adelante.

SQL soporta las generalizaciones mediante la construcción **group by** para realizar las operaciones **cube** y **rollup.** Las construcciones **cube** y **rollup** en la cláusula **group by** permiten que se puedan realizar múltiples consultas **group by** en una única consulta devolviendo el resultado como una única relación en un estilo similar al de la relación de la Figura 5.21.

Considere de nuevo el ejemplo de la tienda y la relación:

ventas(*nombre_elem*, *color*, *talla*, *cantidad*)

Se puede obtener la cantidad de elementos vendidos de cada nombre de elemento escribiendo una simple consulta **group by:**

```
select nombre_elem, sum(cantidad)
from ventas
group by nombre_elem;
```

El resultado de esta consulta se muestra en la Figura 5.23. Observe que representa los mismos datos que los de la última columna de la Figura 5.17 (o de forma equivalente, la primera fila del cubo de la Figura 5.18).

De manera similar, se puede obtener el número de elementos vendidos de cada color, etc. Utilizando varios atributos en la cláusula **group by** es posible obtener cuántos elementos se han vendido con cierto conjunto de propiedades.

Por ejemplo, se puede descubrir una anomalía en las ventas por nombre de elemento y color escribiendo:

```
select nombre_elem, color, sum(cantidad)
from ventas
group by nombre_elem, color;
```

El resultado de esta consulta se muestra en la Figura 5.24. Observe que representa los mismos datos que los de las primeras cuatro columnas de la Figura 5.17 (o de forma equivalente, las cuatro primeras filas y columnas del cubo de la Figura 5.18).

Sin embargo, si se quiere generar el cubo de datos completo usando este enfoque, tendríamos que escribir una consulta separada para cada uno de los siguientes conjuntos de atributos:

{(*nombre_elem*, *color*, *talla*), (*nombre_elem*, *color*), (*nombre_elem*, *talla*), (*color*, *talla*), (*nombre_elem*), (*color*), (*talla*), ()}

donde () indica una lista de **group by** vacía.

La construcción **cube** nos permite conseguirlo en una única consulta:

```
select nombre_elem, color, talla, sum(cantidad)
from ventas
group by cube(nombre_elem, color, talla);
```

Esta consulta genera una relación resultado cuyo esquema es:

(*nombre_elem*, *color*, *talla*, **sum**(*cantidad*))

En consecuencia el resultado de esta consulta es realmente una relación, las tuplas en el resultado contienen *null* como valor de aquellos atributos que no existen en algún agrupamiento. Por ejemplo, las tuplas que se generan al agrupar por *talla* tienen un esquema (*talla*, **sum**(*cantidad*)). Se convierten a tuplas sobre (*nombre_elem*, *color*, *talla*, **sum**(*cantidad*)) insertando *null* en *nombre_elem* y *color*.

Las relaciones de los cubos de datos suelen ser muy grandes. La consulta de cubo anterior, con 3 posibles colores, 4 posibles nombres de elementos y 3 tallas, tiene 80 tuplas. La relación de la Figura 5.21 se genera agrupando por *nombre_elem* y *color*. También usa **all** en lugar de *null* para que resulte más legible al usuario medio. Para generar dicha relación en SQL se sustituyen los **all** por *null*. La consulta:

select *nombre_elem*, *color*, **sum**(*cantidad*)
from *ventas*
group by cube(*nombre_elem*, *color*);

genera la relación de la Figura 5.21 con nulos. La sustitución de **all** se consigue utilizando las funciones **decode** y **grouping** de SQL. La función **decode** es conceptualmente simple pero su sintaxis es un tanto difícil de leer. Consulte el recuadro «la Función Decode» para más detalles.

La construcción **rollup** es la misma que la construcción **cube** excepto que **rollup** genera menos consultas **group by.** Se ha visto que **group by cube** (*nombre_elem*, *color*, *talla*) genera las ocho formas de formar una consulta **group by** utilizando algunos (o todos o ninguno) de los atributos. En:

select *nombre_elem*, *color*, *talla*, **sum**(*cantidad*)
from *ventas*
group by rollup(*nombre_elem*, *color*, *talla*);

group by rollup(*nombre_elem*, *color*, *talla*) genera solo 4 agrupamientos:

{(*nombre_elem*, *color*, *talla*), (*nombre_elem*, *color*), (*nombre_elem*), ()}

Observe que el orden de los atributos en el **rollup** es diferente; el atributo final *talla* en el ejemplo solo aparece en uno de los agrupamientos, el penúltimo (el segundo por el final) en solo dos agrupamientos, y así sucesivamente, apareciendo el primer atributo en todos los agrupamientos excepto en uno (el agrupamiento vacío).

¿Por qué se desearía especificar los agrupamientos que se utilizan en el **rollup**? Estos grupos suelen ser de interés práctico para las jerarquías (como en la Figura 5.19, por ejemplo). Para la jerarquía de ubicación (*Región*, *País*, *Provincia*, *Ciudad*), se puede querer agrupar por *Región* para obtener las ventas por región. Después se puede querer concretar (drill down) al nivel de país dentro de cada región, lo que significa que podíamos agrupar por *Región* y *País*. Concretando más aún, se podría querer agrupar por *Región*, *País* y *Provincia* y después por *Región*, *País*, *Provincia* y *Ciudad*. La construcción **rollup** permite especificar esta secuencia de concreción cada vez de más detalle.

Se pueden utilizar varios **rollup** y **cube** en una única cláusula **group by.** Por ejemplo, la siguiente consulta:

select *nombre_elem*, *color*, *talla*, **sum**(*cantidad*)
from *ventas*
group by rollup(*nombre_elem*), **rollup**(*color*, *talla*);

genera los agrupamientos:

{(*nombre_elem*, *color*, *talla*), (*nombre_elem*, *color*), (*nombre_elem*), (*color*, *talla*), (*color*), ()}

Para entender por qué, observe que **rollup**(*nombre_elem*) genera dos agrupamientos, {(*nombre_elem*), ()}, y **rollup**(*color*, *talla*) genera tres agrupamientos, {(*color*, *talla*), (*color*), ()}. El producto cartesiano de los dos genera los seis agrupamientos mostrados.

Ni la cláusula **rollup** ni **cube** permiten un control sobre los agrupamientos que generan. Por ejemplo, no se pueden utilizar para especificar que solo se desean los agrupamientos {(*color*, *talla*), (*talla*, *nombre_elem*)}. Esta restricción sobre los agrupamientos se puede generar utilizando las construcciones **grouping** en la cláusula **having;** se dejan los detalles como ejercicio para el lector.

nombre_elem	*talla*	*oscuro*	*pastel*	*blanco*
falda	pequeño	2	11	2
falda	mediano	5	9	5
falda	grande	1	15	3
vestido	pequeño	2	4	2
vestido	mediano	6	3	3
vestido	grande	12	3	0
camisa	pequeño	2	4	17
camisa	mediano	6	1	1
camisa	grande	6	2	10
pantalón	pequeño	14	1	3
pantalón	mediano	6	0	0
pantalón	grande	0	1	2

Figura 5.22. Resultado de la operación pivot de SQL sobre la relación *ventas* de la Figura 5.16.

nombre_elem	*cantidad*
falda	53
vestido	35
camisa	49
pantalón	27

Figura 5.23. Resultado de la consulta.

nombre_elem	*color*	*cantidad*
falda	oscuro	8
falda	pastel	35
falda	blanco	10
vestido	oscuro	20
vestido	pastel	10
vestido	blanco	5
camisa	oscuro	14
camisa	pastel	7
camisa	blanco	28
pantalón	oscuro	20
pantalón	pastel	2
pantalón	blanco	5

Figura 5.24. Resultado de la consulta.

LA FUNCIÓN DECODE

La función **decode** permite la sustitución de valores de un atributo de una tupla. La forma general de la función **decode** es:

decode (*valor*, *coincidencia-1*, *sustitución-1*,
coincidencia-2, *sustitución-2*…,
coincidencia-N, *sustitución-N*,*sustitución-predeterminada*);

Compara el *valor* contra los valores de *coincidencia* y si existe algún valor que coincide, sustituye el valor del atributo con el correspondiente valor de sustitución. Si no se produce ninguna coincidencia, entonces el valor del atributo se sustituye por el valor de sustitución predeterminado.

La función **decode** no funciona como desearíamos para los valores nulos, ya que como se ha visto en la Sección 3.6 los predicados sobre nulos se evalúan a **unknown,** que al final se convierte en **false.** Para tratar con ello se aplica la función **grouping,** que devuelve 1 si su argumento es un valor nulo generado por **cube** o **rollup** y 0 en caso contrario. Entonces la relación de la Figura 5.21, con las ocurrencias de **all** sustituidas por *null*, se puede calcular con la consulta:

select decode(**grouping**(*nombre_elem*), 1, ´all´, *nombre_elem*)
as *nombre_elem* **decode**(**grouping**(*color*), 1, ´all´, *color*)
as *color* **sum**(*cantidad*) **as** *cantidad*
from *ventas*
group by cube(*nombre_elem*, *color*);

5.7. Resumen

- Las consultas SQL se pueden invocar desde lenguajes anfitriones mediante SQL incorporado y SQL dinámico. Las normas ODBC and JDBC definen interfaces para programas de aplicación para el acceso a las bases de datos de SQL desde programas en los lenguajes C y Java.

  Los programadores utilizan cada vez más estas API para el acceso a las bases de datos.
- Las funciones y los procedimientos pueden definirse empleando las extensiones procedimentales de SQL que permiten la iteración y las instrucciones condicionales (if-then-else).
- Los disparadores definen las acciones que se deben ejecutar automáticamente cuando se satisfacen las condiciones correspondientes. Los disparadores tienen muchas aplicaciones, como la implementación de reglas de negocio, la auditoría del registro histórico e incluso realizar acciones fuera del sistema de bases de datos.

  Aunque los disparadores solo se han añadido recientemente a la norma de SQL como parte de SQL:1999, la mayor parte de los sistemas de bases de datos los implementan desde hace tiempo.
- Algunas consultas, como el cierre transitivo, pueden expresarse tanto mediante la iteración como mediante las consultas recursivas de SQL. La recursión puede expresarse mediante las vistas recursivas o mediante cláusulas **with** recursivas.
- SQL soporta gran variedad de funcionalidades de agregación avanzadas que incluyen funciones de clasificación y para la creación de ventanas, que simplifican la expresión de algunos agregados y permiten una evaluación más eficiente.
- Las herramientas de procesamiento analítico en línea (online analytical processing, OLAP) ayudan a los analistas a ver los datos resumidos de diferentes maneras, de forma que puedan obtener una perspectiva del funcionamiento de la organización.
  - Las herramientas OLAP trabajan con datos multidimensionales, caracterizados por los atributos de dimensiones y por los atributos de medida.
  - Los cubos de datos constan de datos multidimensionales resumidos de diferentes maneras. El cálculo previo de los cubos ayuda a acelerar las consultas de resúmenes de datos.
  - El formato de las tabulaciones cruzadas permite que los usuarios vean simultáneamente dos dimensiones de los datos multidimensionales, junto con resúmenes de los datos.
  - La concreción, la abstracción y los cortes y creación de datos de los cubos son algunas de las operaciones que los usuarios llevan a cabo con las herramientas OLAP.
- SQL, a partir de la norma SQL:1999, proporciona una variedad de operadores para el análisis de datos, incluyendo las operaciones **cube** y **rollup.** Algunos sistemas soportan la cláusula **pivot,** que permite la creación sencilla de tablas cruzadas.

Términos de repaso

- JDBC.
- ODBC.
- Instrucciones preparadas.
- Acceso a los metadatos.
- Inyección de SQL.
- SQL incorporado.
- Cursores.
- Cursores actualizables.
- SQL dinámico.
- Funciones de SQL.
- Procedimientos almacenados.
- Constructores procedimentales.
- Rutinas de otros lenguajes.
- Disparadores.
- Disparadores antes y después.
- Variables y tablas de transición.
- Consultas recursivas.
- Consultas monótonas.
- Funciones de clasificación.
  - Clasificación.
  - Clasificación densa.
  - División por.
- Ventanas.
- Procesamiento analítico en línea (Online analytical processing, OLAP).
- Datos multidimensionales.
  - Atributos de medida.
  - Atributos de dimensiones.
  - Pivotado.
  - Cubos de datos.
  - Corte (división) y creación de datos.
  - Abstracción y concreción.
- Tablas cruzadas.

Ejercicios prácticos

5.1. Describa las circunstancias en que elegiría utilizar SQL incrustado en lugar de SQL solo, o solamente un lenguaje de propósito general.

5.2. Escriba una función de Java que emplee las características para metadatos de JDBC.

Tome un ResultSet como parámetro de entrada e imprima el resultado en forma tabular, con los nombres correspondientes como encabezados de las columnas.

5.3. Escriba una función de Java que emplee las características para metadatos de JDBC e imprima una lista de todas las relaciones de la base de datos, mostrando para cada relación el nombre y el tipo de datos de los atributos.

5.4. Demuestre cómo forzar la restricción «un profesor no puede enseñar en dos aulas diferentes en un semestre en la misma franja horaria» utilizando un disparador (recuerde que la restricción se puede no cumplir por cambios en la relación *enseña* así como en la relación *sección*).

5.5. Escriba los disparadores para forzar la integridad referencial de *sección* a *franja_horaria*, para actualizaciones de *sección* y de *franja_horaria*. Observe que las que se escribieron en la Figura 5.8 no cubren la operación **update.**

5.6. Para mantener el atributo *tot_créditos* de la relación *estudiante*, realice lo siguiente:

a. Modifique el disparador de actualización de *matricula*, para manejar las actualizaciones que puedan afectar al valor de *tot_créditos*.

b. Escriba un disparador para manejar las inserciones en la relación *matricula*.

c. ¿Bajo qué suposiciones es razonable crear disparadores en la relación *asignatura*?

5.7. Considere la base de datos del banco de la Figura 5.25.

Definimos la vista *sucursal_cliente* de la siguiente forma:

create view *sucursal_cliente* **as**
  select *nombre_sucursal*, *nombre_cliente*
  from *impositor*, *cuenta*
  where *impositor.número_cuenta* = *cuenta.número_cuenta*

Suponga que la vista está *materializada*; es decir, que la vista se calcula y se almacena.

Escriba disparadores para *mantener* la vista, es decir, mantenerla actualizada según las inserciones y los borrados en *impositor* o en *cuenta*. No hay que preocuparse de las actualizaciones.

5.8. Considere la base de datos del banco de la Figura 5.25. Escriba un disparador de SQL que realice la siguiente acción: en la operación **delete** de una cuenta, para cada propietario de la cuenta, comprobar si el propietario tiene más cuentas, y si no tiene más, eliminarlo de la relación *impositor*.

5.9. Indique el modo de expresar **group by cube**(a, b, c, d) utilizando **rollup;** la respuesta solo debe tener una cláusula **group by.**

5.10. Dada la relación *E*(*estudiante*, *asignatura*, *notas*), escriba una consulta de SQL para encontrar los *n* mejores estudiantes según sus notas totales, utilizando la clasificación.

5.11. Considere la relación *ventas* de la Sección 5.6. Escriba una consulta de SQL para calcular la operación cubo para la relación, dada la relación de la Figura 5.21. No hay que utilizar la construcción **cube.**

sucursal(*nombre_sucursal*, *ciudad_sucursal*, *activos*)
cliente(*nombre_cliente*, *calle_cliente*, *ciudad_cliente*)
préstamo(*número_préstamo*, *nombre_sucursal*, *cantidad*)
prestamista(*nombre_cliente*, *número_préstamo*)
cuenta(*número_cuenta*, *nombre_sucursal*, *saldo*)
impositor(*nombre_cliente*, *número_cuenta*)

Figura 5.25. Base de datos del banco para los Ejercicios 5.7, 5.8 y 5.28.

Ejercicios

5.12. Considere la siguiente relación para una base de datos de una compañía:

- emp(enombre, dnombre, sueldo)
- jefe(enombre, jnombre)

y el código Java de la Figura 5.26, que utiliza la API de JDBC. Suponga que los valores de usuarioid, contraseña, nombre de máquina, etc., son correctos. Describa de forma concisa en español qué hace el programa en Java. Es decir, describa con una frase cómo «encuentra al jefe del departamento de juguetes», no una descripción línea a línea de lo que hace el programa en Java.

5.13. Suponga que se le pide que defina una clase MetaImprimir en Java que tenga un método static void imprimeTabla(String r); el método recibe un nombre de relación *r* como entrada, ejecuta la consulta «**select** * **from** *r*» e imprime el resultado en formato de tabla, con los nombres de los atributos en la cabecera de la tabla.

a. ¿Qué necesita conocer sobre la relación *r* para poder imprimir el resultado en el formato tabular indicado?

b. ¿Qué métodos de JDBC le permiten obtener la información solicitada?

c. Escriba el método imprimirTabla(String r) usando la API de JDBC.

5.14. Repita el Ejercicio 5.13 usando ODBC, definiendo void imprimirTabla(char *r) como una función en lugar de como un método.

5.15. Considere una base de datos de empleados con dos relaciones en las que se han subrayado las claves primarias. Escriba una consulta para encontrar las empresas cuyos empleados ganan sueldos más altos, de media, que el sueldo medio de «First Bank Corporation».

empleado(*nombre_empleado*, *calle*, *ciudad*)
trabajo(*nombre_empleado*, *nombre_empresa*, *sueldo*)

a. Empleando las funciones de SQL necesarias.

b. Sin emplear las funciones de SQL.

5.16. Reescriba la consulta de la Sección 5.2.1 que devuelve el nombre y el presupuesto de todos los departamentos con más de doce profesores, usando la cláusula **with** en lugar de la llamada a una función.

5.17. Compare el uso de SQL incrustado con el uso en SQL de funciones definidas en lenguajes de programación de propósito general. ¿En qué circunstancias es preferible utilizar cada una de estas características?

5.18. Modifique la consulta recursiva de la Figura 5.15 para definir la relación:

prerreq_profundidad(*asignatura_id*, *prerreq_id*, *profundidad*)

en la que el atributo *profundidad* indica el número de niveles de prerrequisitos intermedios que hay entre la asignatura y los prerrequisitos. Los prerrequisitos directos tienen una profundidad de 0.

5.19. Considere el esquema relacional:

componente(*id_componente*, *nombre*, *coste*)
subcomponente(*id_componente*, *id_subcomponente*, *cuenta*)

La tupla (c_1, c_2, 3) de la relación subcomponente indica que el componente con identificador de componente c_2 es un subcomponente directo del componente con identificador de componente c_1, y que c_1 tiene tres copias de c_2. Tenga en cuenta que c_2 puede, a su vez, tener más subcomponentes. Escriba una consulta recursiva de SQL que genere el nombre de todos los subcomponentes del componente con identificador «C-100».

5.20. Considere nuevamente el esquema relacional del Ejercicio 5.19. Escriba una función de JDBC que utilice SQL no recursivo para calcular el coste total del componente «C-100», incluido el coste de todos sus subcomponentes. Asegúrese de tener en cuenta el hecho de que cada componente puede tener varios ejemplares de un subcomponente. Se puede utilizar la recursión en Java si se desea.

5.21. Suponga que hay dos relaciones r y s, tales que la clave externa B de r hace referencia a la clave primaria A de s. Describa la manera en que puede usarse el mecanismo de los disparadores para implementar la opción **on delete cascade** cuando se borra una tupla de s.

5.22. La ejecución de un disparador puede hacer que se dispare otra acción. La mayor parte de los sistemas de bases de datos imponen un límite sobre la profundidad máxima de anidamiento. Explique el motivo de la imposición de ese límite.

5.23. Considere la relación r, que se muestra en le Figura 5.27. Indique el resultado de la siguiente consulta:

select *edificio*, *número_aula*, *franja_horaria_id*, **count**(*)
from r
group by rollup (*edificio*, *número_aula*, *franja_horaria_id*)

5.24. Para cada una de las funciones de agregación de SQL **sum, count, min** y **max,** indique el modo de calcular el valor agregado para el multiconjunto $S_1 \cup S_2$, dados los valores agregados para los multiconjuntos S_1 y S_2.

5.25. Basándose en lo anterior, indique las expresiones para calcular los valores agregados con el agrupamiento de un subconjunto S de los atributos de la relación $r(A, B, C, D, E)$, dados los valores agregados para el agrupamiento de los atributos $T \supseteq S$ para las siguientes funciones de agregación:

a. sum, count, min y **max**

b. avg

c. Desviación estándar

En la Sección 5.5.1 se utilizó la vista *estudiante_nota* del Ejercicio 4.5 para escribir una consulta con objeto de encontrar la posición de cada estudiante según su nota media. Modifique la consulta para que solo se muestren los diez primeros estudiantes, es decir, los estudiantes clasificados del 1 al 10.

5.26. Indique un ejemplo de un par de agrupamientos que no puedan expresarse utilizando una única cláusula **group by** con **cube** y con **rollup.**

5.27. Dada la relación $s(a, b, c)$, indique el modo de utilizar las características extendidas de SQL para generar un histograma de c frente a a, dividiendo a en veinte particiones de igual tamaño (es decir, que cada partición contenga el cinco por ciento de las tuplas de r, ordenadas según a).

5.28. Considere la base de datos del banco de la Figura 5.25 y el atributo *saldo* de la relación *cuenta*. Escriba una consulta en SQL para calcular un histograma de los valores de *saldo*, dividiendo el intervalo desde cero hasta el máximo saldo de una cuenta presente, en tres intervalos iguales.

```
import java.sql.*;
public class Misterio {
  public static void main(String[] args) {
    try {
    Connection con=null;
    Class.forName("oracle.jdbc.driver.OracleDriver");
    con=DriverManager.getConnection(
          "jdbc:oracle:thin:star/X@//edgar.cse.lehigh.edu:1521/XE");
    Statement s=con.createStatement();
    String q;
    String empNombre = "dog";
    boolean más;
    ResultSet resultado;
    do {
      q = "select jnombre from jefe where enombre = '" +
            empNombre + "'";
      resultado = s.executeQuery(q);
      más = result.next();
      if (más) {
        empNombre = resultado.getString("jnombre");
        System.out.println(empNombre);
      }
    } while (más);
    s.close();
    con.close();
  } catch(Exception e){e.printStackTrace();} }}
```

Figura 5.26. Código en Java para el Ejercicio 5.12.

edificio	*número_aula*	*franja_horaria_id*	*asignatura_id*	*secc_id*
Garfield	359	A	BIO-101	1
Garfield	359	B	BIO-101	2
Saucon	651	A	CS-101	2
Saucon	550	C	CS-319	1
Painter	705	D	MU-199	1
Painter	403	D	FIN-201	1

Figura 5.27. La relación r para el Ejercicio 5.23.

Notas bibliográficas

Consulte las notas bibliográficas del Capítulo 3 para consultar las referencias a las normas de SQL y a los libros sobre SQL.

java.sun.com/docs/books/tutorial es una excelente fuente de información actualizada sobre JDBC, y sobre Java en general. Las referencias a los libros sobre Java (incluido JDBC) también están disponibles en esta URL. El API de ODBC se describe en Microsoft [1997] y en Sanders [1998]. Melton y Eisenberg [2000] proporcionan una guía de SQLJ, JDBC y las tecnologías relacionadas. Se puede encontrar más información sobre ODBC, ADO y ADO.NET en msdn.microsoft.com/data.

En el contexto de las funciones y procedimientos de SQL, muchos productos de bases de datos soportan características que van más allá de las especificadas en la norma, y no soportan muchas características de la norma. Puede encontrar más información sobre estas características en los manuales de usuario de SQL de los respectivos productos.

Las propuestas originales de asertos y disparadores para SQL se estudian en Astrahan et ál. [1976], Chamberlin et ál. [1976] y Chamberlin et ál. [1981]. Melton y Simon [2001], Melton [2002] y Eisenberg y Melton [1999] ofrecen un tratamiento del nivel de los libros de texto de SQL:1999, la versión de la norma de SQL que primero incluyó los disparadores.

El procesamiento de consultas recursivas se estudió primero en detalle en el contexto del lenguaje de consultas llamado Datalog, que se basaba en la lógica matemática y seguía la sintaxis del lenguaje de programación Prolog. Ramakrishnan y Ullman [1995] proporcionan una revisión de los resultados en esta área, incluyendo las técnicas de optimización de consultas que seleccionan un subconjunto de las tuplas de una vista definida recursivamente.

Gray et ál. [1995] y Gray et ál. [1997] describen el operador cubo de datos. Los algoritmos eficientes para calcular cubos de datos se describen en Agarwal et ál. [1996], Harinarayan et ál. [1996] y Ross y Srivastava [1997]. Las descripciones del soporte extendido de la agregación en SQL:1999 pueden encontrarse en los manuales de los productos de sistemas de bases de datos como Oracle y DB2 de IBM.

Existe una sustancial cantidad de trabajo sobre la eficiencia de las consultas «primeros-k» que devuelven solo los resultados clasificados en los primeros k. Una revisión de estos trabajos aparece en Ilyas et ál. [2008].

Herramientas

La mayor parte de los fabricantes de bases de datos proporcionan herramientas OLAP como parte de sus sistemas de bases de datos, o como aplicaciones complementarias.

Entre ellas están las herramientas OLAP de Microsoft Corp., Oracle Express e Informix Metacube. Las herramientas pueden estar integradas en un producto de «Inteligencia de negocio» como IBM Cognos. Muchas empresas también ofrecen herramientas de análisis para aplicaciones específicas, como la gestión de las relaciones con los clientes (por ejemplo, Oracle Siebel CRM).

Lenguajes formales de consulta relacional

06

En los Capítulos 2 a 5 se introdujo el modelo relacional y se ha tratado SQL con gran nivel de detalle. En este capítulo se presenta el modelo formal en el que se basa SQL, así como otros lenguajes de consulta relacionales.

Se trata de tres lenguajes formales. Se empieza presentando el álgebra relacional, que es la base del ampliamente utilizado lenguaje de consulta SQL. Después se trata el cálculo relacional de tuplas y el cálculo relacional de dominios, que son lenguajes de consulta declarativos basados en lógica matemática.

6.1. El álgebra relacional

El álgebra relacional es un lenguaje de consulta *procedimental*. Consta de un conjunto de operaciones que toman una o dos relaciones de entrada y generan una nueva relación como resultado. Las operaciones fundamentales del álgebra relacional son *selección*, *proyección*, *unión*, *diferencia de conjuntos*, *producto cartesiano* y *renombramiento*. Además de las operaciones fundamentales, existen otras operaciones: *intersección de conjuntos*, *reunión natural* y *asignación*. Estas operaciones se definen en términos de las operaciones fundamentales.

6.1.1. Operaciones fundamentales

Las operaciones selección, proyección y renombramiento se denominan operaciones *unarias* porque operan sobre una sola relación. Las otras tres operaciones operan sobre pares de relaciones y se denominan, por tanto, operaciones *binarias*.

6.1.1.1. Operación selección

La operación **selección** selecciona tuplas que satisfacen un predicado dado. Se usa la letra griega sigma minúscula (σ) para denotar la selección. El predicado aparece como subíndice de σ. La relación de argumentos se da entre paréntesis a continuación de σ. Por tanto, para seleccionar las tuplas de la relación *profesor* en las que el profesor pertenece al departamento de Física se escribe:

$$\sigma_{nombre_dept\ =\ \text{«Física»}}\ (profesor)$$

Si la relación *profesor* es como se muestra en la Figura 6.1, la relación que resulta de la consulta anterior es como aparece en la Figura 6.2.

Se pueden buscar todos los profesores cuyos sueldos sea superior a 90.000 € escribiendo la siguiente consulta:

$$\sigma_{\text{sueldo}\ >\ 90000}\ (profesor)$$

En general, se permiten las comparaciones que usan $=$, $\neq$, $<$, $\leq$, $>$ o $\geq$ en el predicado de selección. Además, se pueden combinar varios predicados en uno mayor usando las conectivas *y* ($\wedge$), *o* ($\vee$) y *no* ($\neg$). Por tanto, para encontrar las tuplas correspondientes a los profesores de Física con un sueldo de más de 90.000 €, se escribe:

$$\sigma_{nombre_dept\ =\ \text{«Física»}\ \wedge\ sueldo\ >\ 90000}\ (profesor)$$

El predicado de selección puede incluir comparaciones entre dos atributos. Para ilustrarlo, considere la relación *departamento*. Para hallar todos los departamentos que se llaman igual que el nombre del edificio se puede escribir:

$$\sigma_{nombre_dept\ =\ edificio}\ (departamento)$$

ID	nombre	nombre_dept	sueldo
10101	Srinivasan	Informática	65000
12121	Wu	Finanzas	90000
15151	Mozart	Música	40000
22222	Einstein	Física	95000
32343	El Said	Historia	60000
33456	Gold	Física	87000
45565	Katz	Informática	75000
58583	Califieri	Historia	62000
76543	Singh	Finanzas	80000
76766	Crick	Biología	72000
83821	Brandt	Informática	92000
98345	Kim	Electrónica	80000

Figura 6.1. La relación *profesor*.

ID	nombre	nombre_dept	sueldo
22222	Einstein	Física	95000
33456	Gold	Física	87000

Figura 6.2. Resultado de $\sigma_{nombre_dept\ =\ \text{«Física»}}\ (profesor)$.

6.1.1.2. Operación proyección

Suponga que se desea obtener una relación de todos los *ID*, *nombres* y *sueldo* de los profesores, pero sin los nombres de departamento. La operación **proyección** permite obtener esa relación. La operación proyección es una operación unaria que devuelve su relación de argumentos, excluyendo algunos atributos. Dado que las relaciones son conjuntos, se eliminan todas las filas duplicadas. La proyección se indica con la letra griega mayúscula pi (Π). Se crea una lista de los atributos que se desea que aparezcan en el resultado como subíndices de Π. Su único argumento, una relación, se escribe a continuación entre paréntesis. La consulta para crear dicha lista puede escribirse como:

$$\Pi_{ID,\ nombre,\ sueldo}\ (profesor)$$

La Figura 6.3 muestra la relación que resulta de esta consulta.

ID	*nombre*	*sueldo*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Figura 6.3. Resultado de $\Pi_{ID,\ nombre,\ sueldo}\ (profesor)$.

RELACIÓN ENTRE SQL Y EL ÁLGEBRA RELACIONAL

El término *selección* del álgebra relacional tiene un significado diferente del que se utiliza en SQL, que resulta de un hecho histórico desafortunado. En álgebra relacional, el término *selección* corresponde a lo que en SQL se denomina *where*. Se va a poner énfasis en la diferente interpretación para minimizar la potencial confusión.

6.1.1.3. Composición de operaciones relacionales

Es importante el hecho de que el resultado de una operación relacional sea también una relación. Considere la consulta más compleja «Buscar el nombre de todos los profesores del departamento de Física». Hay que escribir:

$$\Pi_{name}\ (\sigma_{nombre_dept\ =\ \text{«Física»}}\ (profesor))$$

Téngase en cuenta que, en vez de dar el nombre de una relación como argumento de la operación proyección, se da una expresión cuya evaluación es una relación.

En general, dado que el resultado de las operaciones del álgebra relacional es del mismo tipo (relación) que los datos de entrada, las operaciones del álgebra relacional pueden componerse para formar una **expresión del álgebra relacional.** Componer operaciones del álgebra relacional para formar expresiones del álgebra relacional es igual que componer operaciones aritméticas (como $+$, $-$, $*$ y $\div$) para formar expresiones aritméticas. La definición formal de las expresiones del álgebra relacional se estudia en la Sección 6.1.2.

6.1.1.4. Operación unión

Considere una consulta para buscar el conjunto de todas las asignaturas que se enseñaron en el semestre del otoño de 2009, el semestre de la primavera de 2010 o en ambos. La información se encuentra en la relación *sección* (Figura 6.4). Para encontrar el conjunto de todas las asignaturas que se enseñaron en el semestre del otoño de 2009 se escribe:

$$\Pi_{asignatura_id}\ (\sigma_{semestre\ =\ \text{«Otoño»}\ \wedge\ año\ =\ 2009}\ (sección))$$

Para encontrar el conjunto de todas las asignaturas que se enseñaron en el semestre de la primavera de 2010 se escribe:

$$\Pi_{asignatura_id}\ (\sigma_{semestre\ =\ \text{«Primavera»}\ \wedge\ año\ =\ 2010}\ (sección))$$

Para contestar a la consulta es necesaria la **unión** de estos dos conjuntos; es decir, hacen falta todos los *ID* de sección que aparecen en alguna de las dos relaciones o en ambas. Estos datos se pueden obtener mediante la operación binaria unión, que se indica, como en la teoría de conjuntos, por $\cup$. Por tanto, la expresión buscada es:

$$\Pi_{asignatura_id}\ (\sigma_{semestre\ =\ \text{«Otoño»}\ \wedge\ año\ =\ 2009}\ (sección)) \cup$$
$$\Pi_{asignatura_id}\ (\sigma_{semestre\ =\ \text{«Primavera»}\ \wedge\ año\ =\ 2010}\ (sección))$$

asignatura_id	*secc_id*	*semestre*	*año*	*edificio*	*número_aula*	*franja_horaria_id*
BIO-101	1	Verano	2009	Painter	514	B
BIO-301	1	Verano	2010	Painter	514	A
CS-101	1	Otoño	2009	Packard	101	H
CS-101	1	Primavera	2010	Packard	101	F
CS-190	1	Primavera	2009	Taylor	3128	E
CS-190	2	Primavera	2009	Taylor	3128	A
CS-315	1	Primavera	2010	Watson	120	D
CS-319	1	Primavera	2010	Watson	100	B
CS-319	2	Primavera	2010	Taylor	3128	C
CS-347	1	Otoño	2009	Taylor	3128	A
EE-181	1	Primavera	2009	Taylor	3128	C
FIN-201	1	Primavera	2010	Packard	101	B
HIS-351	1	Primavera	2010	Painter	514	C
MU-199	1	Primavera	2010	Packard	101	D
PHY-101	1	Otoño	2009	Watson	100	A

Figura 6.4. La relación *sección*.

La relación resultante de esta consulta aparece en la Figura 6.5. Tenga en cuenta que en el resultado hay ocho tuplas, aunque haya tres asignaturas distintas ofertadas en el semestre del otoño de 2009 y seis asignaturas distintas ofertadas en el semestre de la primavera de 2010. Como las relaciones son un conjunto, los valores duplicados como CS-101, que se ofrece en ambos semestres, aparecen una sola vez.

Observe que en este ejemplo se toma la unión de dos conjuntos, ambos consistentes en valores de *asignatura_id*. En general, se debe asegurar que las uniones se realicen entre relaciones compatibles. Por ejemplo, no tendría sentido realizar la unión de las relaciones *profesor* y *estudiante*. Aunque ambas tengan cuatro atributos, difieren en los dominios de *sueldo* y *tot_créditos*. La unión de estos dos atributos no tendría sentido en la mayor parte de los casos. Por tanto, para que la operación unión $r \cup s$ sea válida hay que exigir que se cumplan dos condiciones:

1. Las relaciones r y s deben ser de la misma cardinalidad; es decir, deben tener el mismo número de atributos.
2. Los dominios de los atributos i-ésimos de r y de s deben ser iguales para todo i.

Tenga en cuenta que r y s pueden ser, en general, relaciones de la base de datos o relaciones temporales resultado de expresiones del álgebra relacional.

asignatura_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Figura 6.5. Asignaturas ofertadas en los semestres del otoño de 2009 y la primavera de 2010 o en ambos.

6.1.1.5. Operación diferencia de conjuntos

La operación **diferencia de conjuntos,** indicada por $-$, permite encontrar las tuplas que están en una relación pero no en la otra. La expresión $r - s$ da como resultado una relación que contiene las tuplas que están en r pero no en s.

Se pueden buscar todas las asignaturas que se enseñaron en el semestre del otoño de 2009 pero no en el semestre de la primavera de 2010, escribiendo:

$$\Pi_{asignatura_id}\,(\sigma_{semestre\,=\,\text{«Otoño»}\,\wedge\,año\,=\,2009}\,(sección)) -$$

$$\Pi_{asignatura_id}\,(\sigma_{semestre\,=\,\text{«Primavera»}\,\wedge\,año\,=\,2010}\,(sección))$$

La relación resultante de esta consulta aparece en la Figura 6.6.

Como en el caso de la operación unión, hay que asegurarse de que las diferencias de conjuntos se realicen entre relaciones *compatibles*. Por tanto, para que una operación diferencia de conjuntos $r - s$ sea válida se exige que las relaciones r y s sean de la misma cardinalidad y que los dominios de los atributos i-ésimos de r y de s sean iguales, para todo i.

asignatura_id
CS-347
PHY-101

Figura 6.6. Asignaturas que se ofertan en el semestre del otoño de 2009 pero no en el semestre de la primavera de 2010.

6.1.1.6. Operación producto cartesiano

La operación **producto cartesiano,** que se indica mediante un aspa ($\times$), permite combinar información de dos relaciones cualesquiera. El producto cartesiano de las relaciones r_1 y r_2 se escribe $r_1 \times r_2$.

Recuerde que las relaciones se definen como subconjuntos del producto cartesiano de un conjunto de dominios. A partir de esa definición ya debería tener una idea intuitiva sobre la definición de la operación producto cartesiano. Sin embargo, dado que el mismo nombre de atributo puede aparecer tanto en r_1 como en r_2, es necesario crear un convenio de denominación para distinguir unos atributos de otros. En este caso se realiza adjuntando al atributo el nombre de la relación de la que proviene originalmente. Por ejemplo, el esquema de relación $r = profesor \times enseña$ es:

(*profesor.ID*, *profesor.nombre*, *profesor.nombre_dept*, *profesor.sueldo enseña.ID*, *enseña.asignatura_id*, *enseña.secc_id*, *enseña.semestre*, *enseña.año*)

Con este esquema se puede distinguir entre *profesor.ID* y *enseña.ID*. Para los atributos que solo aparecen en uno de los dos esquemas se suele omitir el prefijo con el nombre de la relación. Esta simplificación no genera ambigüedad alguna. Por tanto, se puede escribir el esquema de la relación r como:

(*profesor.ID*, *nombre*, *nombre_dept*, *sueldo enseña.ID*, *asignatura_id*, *secc_id*, *semestre*, *año*)

Este convenio de denominaciones exige que las relaciones que sean argumentos de la operación producto cartesiano tengan nombres diferentes. Esta exigencia causa problemas en algunos casos, como cuando se desea calcular el producto cartesiano de una relación consigo misma. Se produce un problema parecido si se usa el resultado de una expresión del álgebra relacional en un producto cartesiano, dado que hará falta un nombre de relación para poder hacer referencia a sus atributos. En la Sección 6.1.1.7 se verá la manera de evitar estos problemas mediante la operación renombramiento.

Ahora que se conoce el esquema de relación de $r = profesor \times enseña$ es necesario encontrar las tuplas que aparecerán en r. Como es posible imaginar, se crea una tupla de r a partir de cada par de tuplas posible: una de la relación *profesor* (Figura 6.1) y otra de *enseña* (Figura 6.7). Por tanto, r es una relación de gran tamaño, como se puede ver en la Figura 6.8, en la que solo se ha incluido una parte de las tuplas que constituyen r.[1]

ID	asignatura_id	secc_id	semestre	año
10101	CS-101	1	Otoño	2009
10101	CS-315	1	Primavera	2010
10101	CS-347	1	Otoño	2009
12121	FIN-201	1	Primavera	2010
15151	MU-199	1	Primavera	2010
22222	PHY-101	1	Otoño	2009
32343	HIS-351	1	Primavera	2010
45565	CS-101	1	Primavera	2010
45565	CS-319	1	Primavera	2010
76766	BIO-101	1	Verano	2009
76766	BIO-301	1	Verano	2010
83821	CS-190	1	Primavera	2009
83821	CS-190	2	Primavera	2009
83821	CS-319	2	Primavera	2010
98345	EE-181	1	Primavera	2009

Figura 6.7. La relación *enseña*.

1 Observe que se ha renombrado *profesor.ID* como *prof.ID* para reducir el ancho de las tablas de las Figuras 6.8 y 6.9.

prof.ID	*nombre*	*nombre_dept*	*sueldo*	*enseña.ID*	*asignatura_id*	*secc_id*	*semestre*	*año*
10101	Srinivasan	Física	95000	10101	CS-101	1	Otoño	2009
10101	Srinivasan	Física	95000	10101	CS-315	1	Primavera	2010
10101	Srinivasan	Física	95000	10101	CS-347	1	Otoño	2009
10101	Srinivasan	Física	95000	10101	FIN-201	1	Primavera	2010
10101	Srinivasan	Física	95000	15151	MU-199	1	Primavera	2010
10101	Srinivasan	Física	95000	22222	PHY-101	1	Otoño	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Física	95000	10101	CS-101	1	Otoño	2009
12121	Wu	Física	95000	10101	CS-315	1	Primavera	2010
12121	Wu	Física	95000	10101	CS-347	1	Otoño	2009
12121	Wu	Física	95000	10101	FIN-201	1	Primavera	2010
12121	Wu	Física	95000	15151	MU-199	1	Primavera	2010
12121	Wu	Física	95000	22222	PHY-101	1	Otoño	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Física	95000	10101	CS-101	1	Otoño	2009
15151	Mozart	Física	95000	10101	CS-315	1	Primavera	2010
15151	Mozart	Física	95000	10101	CS-347	1	Otoño	2009
15151	Mozart	Física	95000	10101	FIN-201	1	Primavera	2010
15151	Mozart	Física	95000	15151	MU-199	1	Primavera	2010
15151	Mozart	Física	95000	22222	PHY-101	1	Otoño	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Física	95000	10101	CS-101	1	Otoño	2009
22222	Einstein	Física	95000	10101	CS-315	1	Primavera	2010
22222	Einstein	Física	95000	10101	CS-347	1	Otoño	2009
22222	Einstein	Física	95000	10101	FIN-201	1	Primavera	2010
22222	Einstein	Física	95000	15151	MU-199	1	Primavera	2010
22222	Einstein	Física	95000	22222	PHY-101	1	Otoño	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Figura 6.8. Resultado de $profesor \times enseña$.

prof.ID	*nombre*	*nombre_dept*	*sueldo*	*enseña.ID*	*asignatura_id*	*secc_id*	*semestre*	*año*
22222	Einstein	Física	95000	10101	CS-437	1	Otoño	2009
22222	Einstein	Física	95000	10101	CS-315	1	Primavera	2010
22222	Einstein	Física	95000	12121	FIN-201	1	Primavera	2010
22222	Einstein	Física	95000	15151	MU-199	1	Primavera	2010
22222	Einstein	Física	95000	22222	PHY-101	1	Otoño	2009
22222	Einstein	Física	95000	32343	HIS-351	1	Primavera	2010
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
33456	Gold	Física	87000	10101	CS-437	1	Otoño	2009
33456	Gold	Física	87000	10101	CS-315	1	Primavera	2010
33456	Gold	Física	87000	12121	FIN-201	1	Primavera	2010
33456	Gold	Física	87000	15151	MU-199	1	Primavera	2010
33456	Gold	Física	87000	22222	PHY-101	1	Otoño	2009
33456	Gold	Física	87000	32343	HIS-351	1	Primavera	2010
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Figura 6.9. Resultado de $\sigma_{nombre_dept\ =\ \text{«Física»}}\ (profesor \times enseña)$.

Suponga que se tienen n_1 tuplas en *profesor* y n_2 tuplas en *enseña*. Por tanto, hay $n_1 * n_2$ maneras de escoger un par de tuplas, una tupla de cada relación; por lo que hay $n_1 * n_2$ tuplas en r. En concreto, observe que para algunas tuplas t de r puede ocurrir que $t[profesor.ID] \neq t[enseña.ID]$.

En general, si se tienen las relaciones $r_1(R_1)$ y $r_2(R_2)$, $r_1 \times r_2$ es una relación cuyo esquema es la concatenación de R_1 y de R_2. La relación R contiene todas las tuplas t para las que existe una tupla t_1 en r_1 y una tupla t_2 en r_2 cumpliéndose que $t[R_1] = t_1[R_1]$ y $t[R_2] = t_2[R_2]$.

Suponga que se desea determinar el nombre de todos los profesores del departamento de Física junto con los *asignatura_id* de todas las asignaturas que se enseñan; para ello se necesita información de las relaciones *profesor* y *enseña*. Si se escribe:

$$\sigma_{nombre_dept\ =\ \text{«Física»}}\ (profesor \times enseña)$$

entonces el resultado es la relación de la Figura 6.9.

Se tiene una relación que solo pertenece a los profesores del departamento de Física. Sin embargo, la columna *asignatura_id* puede contener información de asignaturas que no se enseñaron por el profesor correspondiente. (Si no se ve el motivo por el que es cierto, recuerde que el producto cartesiano toma todos los emparejamientos posibles de cada tupla de *profesor* con cada tupla de *enseña*).

Dado que la operación producto cartesiano asocia *todas* las tuplas de *profesor* con todas las tuplas de *enseña*, se sabe que, si hay un profesor del departamento de Física y ha enseñado alguna asignatura (como aparece en la relación *enseña*), entonces existe alguna tupla en $\sigma_{nombre_dept\ =\ \text{«Física»}}\ (profesor \times enseña)$ con su nombre que satisface que $profesor.ID = enseña.ID$. Por tanto, se escribe:

$$\sigma_{profesor.ID\ =\ enseña.ID}\ (\sigma_{nombre_dept\ =\ \text{«Física»}}\ (profesor \times enseña))$$

solo se obtienen las tuplas de profesor × enseña que corresponden a los profesores de Física y las asignaturas que enseñan.

Finalmente, dado que solo se desea obtener los nombres de todos los profesores del departamento de Física, junto con los *asignatura_id* de todas las asignaturas que enseñan, se hace una proyección:

$$\Pi_{nombre,\ asignatura_id}\ (\sigma_{profesor.ID\ =\ enseña.ID}\ (\sigma_{nombre_dept\ =\ \text{«Física»}}\ (profesor \times enseña)))$$

El resultado de esta expresión, mostrada en la Figura 6.10, es la respuesta correcta a la consulta formulada. Observe que aunque el profesor Gold pertenece al departamento de Física, no enseña ninguna asignatura (como aparece en la relación *enseña*) y, por tanto, no aparece en el resultado.

nombre	*asignatura_id*
Einstein	PHY-101

Figura 6.10. Resultado de:

$$\Pi_{nombre,\ asignatura_id}\ (\sigma_{profesor.ID\ =\ enseña.ID}\ (\sigma_{nombre_dept\ =\ \text{«Física»}}\ (profesor \times enseña))).$$

Observe que habitualmente existe más de una forma de escribir una consulta en álgebra relacional. Considere la siguiente consulta:

$$\Pi_{nombre,\ asignatura_id}\ (\sigma_{profesor.ID\ =\ enseña.ID}\ ((\sigma_{nombre_dept\ =\ \text{«Física»}}\ (profesor)) \times enseña))$$

Fíjese en la sutil diferencia entre las dos consultas: en la consulta anterior, la selección que restringe *nombre_dept* al departamento de Física se aplica a *profesor*, y el producto cartesiano se aplica después; por el contrario, en la primera consulta el producto cartesiano se aplicaba antes de la selección. Sin embargo, las dos consultas son **equivalentes,** es decir, generan el mismo resultado sobre cualquier base de datos.

6.1.1.7. Operación renombramiento

A diferencia de las relaciones de la base de datos, los resultados de las expresiones del álgebra relacional no tienen un nombre que se pueda usar para referirse a ellas. Resulta útil poder ponerles nombre; la operación **renombramiento,** que se indica con la letra griega ro minúscula (ρ), permite hacerlo. Dada una expresión E del álgebra relacional, la expresión:

$$\rho_x\ (E)$$

devuelve el resultado de la expresión E con el nombre x.

Las relaciones r, por sí mismas, se consideran expresiones (triviales) del álgebra relacional. Por tanto, también se puede aplicar la operación renombramiento a una relación r para obtener la misma relación con un nombre nuevo.

Una segunda forma de la operación renombramiento es la siguiente. Suponga que una expresión del álgebra relacional E tiene cardinalidad n. Entonces, la expresión:

$$\rho_{x\ (A_1,\ A_2,\cdots\ A_n)}\ (E)$$

devuelve el resultado de la expresión E con el nombre x y con los atributos con el nombre cambiado a $A_1, A_2, \ldots A_n$.

Para ilustrar el renombramiento de relaciones, considere la consulta «Encontrar el sueldo más alto de la universidad». La estrategia empleada consiste en: (1) calcular en primer lugar una relación temporal con los sueldos que *no* sean los más altos y (2) realizar la diferencia entre la relación $\Pi_{sueldo}(profesor)$ y la relación temporal recién calculada anteriormente, para obtener el resultado.

1. Paso 1: para calcular la relación temporal hay que comparar los valores de los sueldos de todas las cuentas. Esta comparación se hará calculando el producto cartesiano *profesor* × *profesor* y formando una selección para comparar el valor de dos sueldos cualesquiera que aparezcan en una tupla. En primer lugar hay que crear un mecanismo para distinguir entre los dos atributos sueldo. Se usará la operación renombramiento para cambiar el nombre de una referencia a la relación *profesor*; de este modo se puede hacer referencia dos veces a la relación sin ambigüedad alguna.

sueldo
65000
90000
40000
60000
87000
75000
62000
72000
80000
92000

Figura 6.11. Resultado de la subexpresión:

$$\Pi_{profesor.sueldo}\ (\sigma_{profesor.sueldo\ <\ d.sueldo}\ (profesor \times \rho_d\ (profesor))).$$

Ahora se puede escribir la relación temporal que se compone de los sueldos que no sean los más altos, como:

$$\Pi_{profesor.sueldo}\ (\sigma_{profesor.sueldo\ <\ d.sueldo}\ (profesor \times \rho_d\ (profesor)))$$

Esta expresión proporciona los sueldos de la relación *profesor* para los que aparece un sueldo mayor en alguna parte de la relación *profesor* (cuyo nombre se ha renombrado a d). El resultado contiene todos los sueldos *excepto* el más alto. La Figura 6.11 muestra esta relación.

2. Paso 2: la consulta para determinar el sueldo máximo de la universidad se puede escribir de la siguiente manera:

$$\Pi_{sueldo}(profesor) - \Pi_{profesor.sueldo}(\sigma_{profesor.sueldo < d.sueldo}(profesor \times \rho_d(profesor)))$$

La Figura 6.12 muestra el resultado de esta consulta.

La operación renombramiento no es estrictamente necesaria, dado que es posible usar una notación posicional para los atributos. Se pueden nombrar los atributos de una relación de manera implícita, donde 1 €, 2 €,… hagan referencia, respectivamente, al primer atributo, al segundo, etc. La notación posicional también se aplica a los resultados de las operaciones del álgebra relacional. La siguiente expresión del álgebra relacional muestra el empleo de esta notación posicional para escribir la expresión vista anteriormente, que calcula los salarios que no sean los más altos:

$$\Pi_{\$4}(\sigma_{\$4 < \$8}(profesor \times profesor))$$

Adviértase que el producto cartesiano concatena los atributos de las dos relaciones. Por tanto, para el resultado del producto cartesiano (*profesor* × *profesor*), 4 € se refiere al atributo *sueldo* de la primera ocurrencia de *profesor*, mientras que 8 € se refiere al atributo *sueldo* de la segunda ocurrencia de *profesor*. La notación posicional también se puede utilizar para referirse a los nombres de la relación si una operación binaria necesita distinguir entre las dos relaciones. Por ejemplo, *R*1 € puede hacer referencia a la primera relación del operador, y *R*2 € a la segunda de un producto cartesiano. Sin embargo, la notación posicional no resulta conveniente para las personas, dado que la posición del atributo es un número en vez de un nombre de atributo, que resulta más fácil de recordar. Por tanto, en este libro no se usa la notación posicional.

6.1.2. Definición formal del álgebra relacional

Las operaciones de la Sección 6.1.1 permiten dar una definición completa de las expresiones del álgebra relacional. Las expresiones fundamentales del álgebra relacional se componen de alguno de los siguientes elementos:

- Una relación de la base de datos.
- Una relación constante.

Las relaciones constantes se escriben poniendo una relación de sus tuplas entre llaves { }, por ejemplo {(22222, Einstein, Física, 95000), (76543, Singh, Finanzas, 80000)}.

Las expresiones generales del álgebra relacional se construyen a partir de subexpresiones más pequeñas. Sean E_1 y E_2 expresiones del álgebra relacional. Todas las expresiones siguientes son también expresiones del álgebra relacional:

- $E_1 \cup E_2$.
- $E_1 - E_2$.
- $E_1 \times E_2$.
- $\sigma_P(E_1)$, donde P es un predicado sobre los atributos de E_1.
- $\Pi_S(E_1)$, donde S es una lista que se compone de algunos de los atributos de E_1.
- $\rho_x(E_1)$, donde x es el nuevo nombre del resultado de E_1.

6.1.3. Otras operaciones del álgebra relacional

Las operaciones fundamentales del álgebra relacional son suficientes para expresar cualquier consulta del álgebra relacional. Sin embargo, limitándose exclusivamente a las operaciones fundamentales, algunas consultas habituales resultan complicadas de expresar. Por tanto, se definen otras operaciones que no añaden potencia al álgebra, pero que simplifican las consultas habituales. Para cada operación nueva se facilita una expresión equivalente usando solo las operaciones fundamentales.

6.1.3.1. Operación intersección de conjuntos

La primera operación adicional del álgebra relacional que se va a definir es la **intersección de conjuntos** (∩). Suponga que se desea encontrar el conjunto de todas las asignaturas que se enseñan tanto en el semestre del otoño de 2009 como en el semestre de la primavera de 2010. Empleando la intersección de conjuntos, se puede escribir:

$$\Pi_{asignatura_id}(\sigma_{semestre = \text{«Otoño»} \wedge año = 2009}(sección)) \cap \Pi_{asignatura_id}(\sigma_{semestre = \text{«Primavera»} \wedge año = 2010}(sección))$$

La relación resultante de esta consulta aparece en la Figura 6.13.

Observe que se puede volver a escribir cualquier expresión del álgebra relacional que utilice la intersección de conjuntos sustituyendo la operación intersección por un par de operaciones de diferencia de conjuntos, de la siguiente manera:

$$r \cap s = r - (r - s)$$

Por tanto, la intersección de conjuntos no es una operación fundamental y no añade potencia al álgebra relacional. Sencillamente, es más conveniente escribir $r \cap s$ que $r - (r - s)$.

sueldo
95000

Figura 6.12. Sueldo más alto de la universidad.

asignatura_id
CS-101

Figura 6.13. Asignaturas que se ofertan en el semestre del otoño de 2009 y en el semestre de la primavera de 2010.

6.1.3.2. Operación reunión natural

Suele resultar deseable simplificar ciertas consultas que exijan un producto cartesiano. Generalmente, las consultas que implican un producto cartesiano incluyen un operador selección sobre el resultado del producto cartesiano. La operación de selección más común suele requerir que todos los atributos que sean comunes en las dos relaciones del producto cartesiano tengan los mismos valores.

En el ejemplo de la Sección 6.1.1.6, que combinaba la información de las tablas *profesor* y *enseña*, la condición de coincidencia requería que *profesor.ID* fuese igual que *enseña.ID*. Estos son los únicos dos atributos de las relaciones que tenían el mismo nombre.

La *reunión natural* es una operación binaria que permite combinar ciertas selecciones y un producto cartesiano en una sola operación. Se indica con el símbolo de **reunión** ⋈. La operación reunión natural forma un producto cartesiano de sus dos argumentos, realiza una selección forzando la igualdad de los atributos que aparecen en ambos esquemas de la relación y, finalmente, elimina los atributos duplicados. Volviendo al ejemplo de las relaciones *profesor* y *enseña*, el cálculo de *profesor* **reunión natural** *enseña* solo considera aquellos pares de tuplas en los que las tuplas de *profesor* y las tuplas de *enseña* tienen el mismo valor en el atributo común *ID*. La relación resultado, que se muestra en la Figura 6.14, solo tiene trece tuplas, las que proporcionan la información sobre los profesores y las asignaturas que el profesor enseña realmente. Fíjese que no se repiten los atributos que aparecen en el esquema de las dos relaciones, apareciendo solo una vez. Observe también el orden en que se listan los atributos: primero los atributos comunes de los esquemas de ambas relaciones, después los atributos únicos del esquema de la primera relación y, finalmente, aquellos atributos únicos del esquema de la segunda relación.

ID	nombre	nombre_dept	sueldo	asignatura_id	secc_id	semestre	año
10101	Srinivasan	Informática	65000	CS-101	1	Otoño	2009
10101	Srinivasan	Informática	65000	CS-315	1	Primavera	2010
10101	Srinivasan	Informática	65000	CS-347	1	Otoño	2009
12121	Wu	Finanzas	90000	FIN-201	1	Primavera	2010
15151	Mozart	Música	40000	MU-199	1	Primavera	2010
22222	Einstein	Física	95000	PHY-101	1	Otoño	2009
32343	El Said	Historia	60000	HIS-351	1	Primavera	2010
45565	Katz	Informática	75000	CS-101	1	Primavera	2010
45565	Katz	Informática	75000	CS-319	1	Primavera	2010
76766	Crick	Biología	72000	BIO-101	1	Verano	2009
76766	Crick	Biología	72000	BIO-301	1	Verano	2010
83821	Brandt	Informática	92000	CS-190	1	Primavera	2009
83821	Brandt	Informática	92000	CS-190	2	Primavera	2009
83821	Brandt	Informática	92000	CS-319	2	Primavera	2010
98345	Kim	Electrónica	80000	EE-181	1	Primavera	2009

Figura 6.14. Reunión natural de la relación *profesor* con la relación *enseña*.

Aunque la definición de la reunión natural es compleja, la operación es sencilla de aplicar. A modo de ejemplo, considere nuevamente el ejemplo «Encontrar los nombres de todos los profesores y los *asignatura_id* de todas las asignaturas que enseñan». Esta consulta se puede expresar usando la reunión natural de la siguiente manera:

$$\Pi_{nombre,\ asignatura_id}\,(profesor \bowtie enseña)$$

Dado que los esquemas de *profesor* y de *enseña* tienen en común el atributo *ID*, la operación reunión natural solo considera los pares de tuplas que tienen el mismo valor de *ID*. Esta operación combina cada uno de estos pares en una sola tupla en la unión de los dos esquemas (es decir, *ID*, *nombre*, *nombre_dept*, *sueldo*, *asignatura_id*). Después de realizar la proyección, se obtiene la relación mostrada en la Figura 6.15.

Considere dos esquemas de relación R y S que son, por supuesto, listas de nombres de atributos. Si se consideran los esquemas como *conjuntos*, en vez de como listas, se pueden indicar los nombres de los atributos que aparecen tanto en R como en S como $R \cap S$, y los nombres de los atributos que aparecen en R, en S o en ambos, con $R \cup S$. De manera parecida, los nombres de los atributos que aparecen en R pero no en S se indican con $R - S$, mientras que $S - R$ indica los nombres de los atributos que aparecen en S pero no en R. Observe que las operaciones unión, intersección y diferencia aquí operan sobre conjuntos de atributos, y no sobre relaciones.

Ahora es posible dar una definición formal de la reunión natural. Considere dos relaciones $r(R)$ y $s(S)$. La **reunión natural** de r y de s, que se indica con $r \bowtie s$, es una relación del esquema $R \cup S$ definida formalmente de la siguiente manera:

$$r \bowtie s = \Pi_{R \cup S}\,(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \cdots \wedge r.A_n = s.A_n}\,(r \times s))$$

donde $R \cap S = \{A_1, A_2, \ldots A_n\}$.

Tenga en cuenta que si r(R) y s(S) son relaciones sin ningún atributo en común, es decir, $R \cap S = \varnothing$, entonces $r \bowtie s = r \times s$.

Veamos un ejemplo más del uso de la reunión natural, para escribir la consulta «Encontrar los nombres de todos los profesores del departamento de Informática junto con los asignatura_id de todas las asignaturas que enseñan los profesores».

Como la reunión natural es fundamental para gran parte de la teoría y de la práctica de las bases de datos relacionales, se ofrecen varios ejemplos de su uso:

$$\Pi_{nombre,nombre_asig}\,(\sigma_{nombre_dept\ =\ \text{«Informática»}}\,(profesor \bowtie enseña \bowtie asignatura))$$

La relación resultante de esta consulta aparece en la Figura 6.16.

Observe que se ha escrito $profesor \bowtie enseña \bowtie asignatura$ sin añadir paréntesis para especificar el orden en el que se deben ejecutar las operaciones reunión natural sobre las tres relaciones. En el caso anterior hay dos posibilidades:

$$(profesor \bowtie enseña) \bowtie asignatura$$
$$profesor \bowtie (enseña \bowtie asignatura)$$

No se ha especificado la expresión deseada, ya que las dos son equivalentes. Es decir, la reunión natural es **asociativa.**

La operación *reunión zeta* es una extensión de la operación reunión natural que permite combinar una selección y un producto cartesiano en una sola operación. Considere las relaciones $r(R)$ y $s(S)$, y sea θ un predicado de los atributos del esquema $R \cup S$. La operación **reunión zeta** $r \bowtie_\theta s$ se define de la siguiente manera:

$$r \bowtie_\theta s = \sigma_\theta\,(r \times s)$$

nombre	asignatura_id
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

Figura 6.15. Resultado de $\Pi_{nombre,\ asignatura_id}\,(profesor \bowtie enseña)$.

nombre	nombre_asig
Brandt	Diseño de juegos
Brandt	Procesamiento de imágenes
Katz	Procesamiento de imágenes
Katz	Intro. a la Informática
Srinivasan	Intro. a la Informática
Srinivasan	Robótica
Srinivasan	Conceptos de bases de datos

Figura 6.16. Resultado de:
$\Pi_{nombre,nombre_asig}\,(\sigma_{nombre_dept\ =\ \text{«Informática»}}\,(profesor \bowtie enseña \bowtie asignatura))$.

6.1.3.3. Operación asignación

En ocasiones resulta conveniente escribir una expresión del álgebra relacional mediante la asignación de partes de esa expresión a variables de relación temporal. La operación **asignación,** que se indica con $\leftarrow$, actúa de manera parecida a la asignación de los lenguajes de programación. Para ver el funcionamiento de esta operación, considere la definición de la reunión natural. Se puede escribir $r \bowtie s$ como:

$$temp1 \leftarrow R \times S$$
$$temp2 \leftarrow \sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \cdots \wedge r.A_n = s.A_n}(temp1)$$
$$resultado = \Pi_{R \cup S}(temp2)$$

La evaluación de una asignación no hace que se muestre ninguna relación al usuario. Por el contrario, el resultado de la expresión situada a la derecha de $\leftarrow$ se asigna a la variable relación situada a la izquierda de $\leftarrow$. Esta variable relación puede usarse en expresiones posteriores.

Con la operación asignación se pueden escribir las consultas como programas secuenciales que constan de una serie de asignaciones seguidas de una expresión cuyo valor se muestra como resultado de la consulta. En las consultas del álgebra relacional la asignación siempre debe hacerse a una variable de relación temporal. Las asignaciones a relaciones permanentes constituyen una modificación de la base de datos. Observe que la operación asignación no añade potencia alguna al álgebra. Resulta, sin embargo, una manera conveniente de expresar las consultas complejas.

6.1.3.4. Reunión externa

La operación **reunión externa** es una extensión de la operación reunión para trabajar con información ausente. Suponga que existe algún profesor que no enseña ninguna asignatura. Entonces, la tupla en la relación *profesor* (Figura 6.1) para ese profesor concreto no satisface la condición de la reunión natural con la relación *enseña* (Figura 6.7) y los datos de dicho profesor no aparecerían en el resultado de la reunión natural, como se muestra en la Figura 6.14. Por ejemplo, los profesores Califieri, Gold y Singh no aparecen en el resultado de la reunión natural, ya que no enseñan ninguna asignatura.

De forma más general, algunas tuplas que se encuentren en una de las dos partes de las relaciones en las que se realiza la reunión se pueden «perder». La operación de **reunión externa** funciona de forma similar a la operación de reunión natural que ya se ha visto, pero conserva las tuplas que se perderían al realizar la reunión creando tuplas en el resultado con valores nulos.

Se puede usar la operación *reunión externa* para evitar esta pérdida de información. En realidad, esta operación tiene tres formas diferentes: *reunión externa por la izquierda*, que se indica con $⟕$; *reunión externa por la derecha*, que se indica con $⟖$ y *reunión externa completa*, que se indica con $⟗$. Las tres formas de la reunión externa calculan la reunión y añaden tuplas adicionales al resultado de la misma. Por ejemplo, el resultado de las expresiones *profesor* $⟕$ *enseña* y *enseña* $⟖$ *profesor* se muestra en las Figuras 6.17 y 6.18, respectivamente.

La **reunión externa por la izquierda** ($⟕$) toma todas las tuplas de la relación de la izquierda que no coinciden con ninguna tupla de la relación de la derecha, las rellena con valores nulos en todos los demás atributos de la relación de la derecha y las añade al resultado de la reunión natural. En la Figura 6.17 la tupla (58583, Califieri, Historia, 62000, *null*, *null*, *null*, *null*) es una tupla de este tipo. Toda la información de la relación de la izquierda está en el resultado de la reunión externa por la izquierda.

La **reunión externa por la derecha** ($⟖$) es simétrica a la reunión externa por la izquierda. Rellena con valores nulos las tuplas de la relación de la derecha que no coinciden con ninguna tupla de la relación de la izquierda y las añade al resultado de la reunión natural. En la Figura 6.18 la tupla (58583, *null*, *null*, *null*, *null*, Califieri, Historia, 62000) es una tupla de este tipo. Por tanto, toda la información de la relación de la derecha está en el resultado de la reunión externa por la derecha.

La **reunión externa completa** ($⟗$) realiza estas dos operaciones, rellenando las tuplas de la relación de la izquierda que no coinciden con ninguna tupla de la relación de la derecha y las tuplas de la relación de la derecha que no coinciden con ninguna tupla de la relación de la izquierda y las añade al resultado de la reunión.

Fíjese que para pasar del ejemplo de la reunión externa por la izquierda al ejemplo de la reunión externa por la derecha se ha elegido intercambiar el orden de los operandos. Por tanto, ambos ejemplos preservan las tuplas de la relación *profesor* y contienen la misma información. En el ejemplo, las tuplas de *enseña* siempre tienen tuplas coincidentes en *profesor* y, por tanto, *enseña* $⟕$ *profesor* generaría el mismo resultado que *enseña* $\bowtie$ *profesor*. Si hubiese tuplas en la relación *enseña* que no coincidiesen con tuplas de la relación *profesor*, estas tuplas aparecerían rellenas con nulos en *enseña* $⟕$ *profesor* y en *enseña* $⟖$ *profesor*. Puede encontrar más ejemplo de reuniones externas (expresadas con la sintaxis de SQL) en la Sección 4.1.2.

Como las operaciones de reunión externa pueden generar resultados que contengan valores nulos, es necesario especificar la manera en que deben manejar estos valores las diferentes operaciones del álgebra relacional. La Sección 3.6 aborda este problema. Los mismos conceptos son aplicables al álgebra relacional, por lo que se omiten los detalles.

Es interesante observar que las operaciones de reunión externa pueden expresarse mediante las operaciones básicas del álgebra relacional. Por ejemplo, la operación de reunión externa por la izquierda $r ⟕ s$ se puede expresar como:

$$(r \bowtie s) \cup (r - \Pi_R(r \bowtie s)) \times \{(null, \ldots null)\}$$

donde la relación constante $\{(null, \ldots null)\}$ se encuentra en el esquema $S - R$.

6.1.4. Operaciones del álgebra relacional extendida

En esta sección se describen operaciones del álgebra relacional que proporcionan la capacidad de escribir consultas que no se pueden expresar con las operaciones básicas del álgebra relacional. Estas operaciones se denominan operaciones del **álgebra relacional extendida.**

6.1.4.1. Proyección generalizada

La primera operación es la **proyección generalizada,** que extiende la proyección permitiendo que se utilicen operaciones aritméticas o de cadenas de caracteres en la lista de proyección. La operación proyección generalizada es de la forma:

$$\Pi_{F_1, F_2, \cdots F_n}(E)$$

donde E es cualquier expresión del álgebra relacional y $F_1, F_2, \ldots F_n$ son expresiones con constantes y atributos del esquema de E. Como caso base, las expresiones pueden ser simplemente un atributo o una constante. En general, las expresiones pueden usar operaciones aritméticas como $+$, $-$, $*$ y $\div$ sobre atributos de valor numérico, sobre constantes numéricas y sobre expresiones que generen un resultado numérico. La proyección generalizada también permite operaciones sobre otros tipos de datos, como la concatenación de cadenas de caracteres.

Por ejemplo, la expresión:

$$\Pi_{ID,\ nombre,\ nombre_dept,\ sueldo \div 12}(profesor)$$

devuelve el *ID*, *nombre*, *nombre_dept* y el sueldo mensual de los profesores.

ID	*nombre*	*nombre_dept*	*sueldo*	*asignatura_id*	*secc_id*	*semestre*	*año*
10101	Srinivasan	Informática	65000	CS-101	1	Otoño	2009
10101	Srinivasan	Informática	65000	CS-315	1	Primavera	2010
10101	Srinivasan	Informática	65000	CS-347	1	Otoño	2009
12121	Wu	Finanzas	90000	FIN-201	1	Primavera	2010
15151	Mozart	Música	40000	MU-199	1	Primavera	2010
22222	Einstein	Física	95000	PHY-101	1	Otoño	2009
32343	El Said	Historia	60000	HIS-351	1	Primavera	2010
33456	Gold	Física	87000	*null*	*null*	*null*	*null*
45565	Katz	Informática	75000	CS-101	1	Primavera	2010
45565	Katz	Informática	75000	CS-319	1	Primavera	2010
58583	Califieri	Historia	62000	*null*	*null*	*null*	*null*
76543	Singh	Finanzas	80000	*null*	*null*	*null*	*null*
76766	Crick	Biología	72000	BIO-101	1	Verano	2009
76766	Crick	Biología	72000	BIO-301	1	Verano	2010
83821	Brandt	Informática	92000	CS-190	1	Primavera	2009
83821	Brandt	Informática	92000	CS-190	2	Primavera	2009
83821	Brandt	Informática	92000	CS-319	2	Primavera	2010
98345	Kim	Electrónica	80000	EE-181	1	Primavera	2009

Figura 6.17. Resultado de *profesor* ⟕ *enseña*.

ID	*asignatura_id*	*secc_id*	*semestre*	*año*	*nombre*	*nombre_dept*	*sueldo*
10101	CS-101	1	Otoño	2009	Srinivasan	Informática	65000
10101	CS-315	1	Primavera	2010	Srinivasan	Informática	65000
10101	CS-347	1	Otoño	2009	Srinivasan	Informática	65000
12121	FIN-201	1	Primavera	2010	Wu	Finanzas	90000
15151	MU-199	1	Primavera	2010	Mozart	Música	40000
22222	PHY-101	1	Otoño	2009	Einstein	Física	95000
32343	HIS-351	1	Primavera	2010	El Said	Historia	60000
33456	*null*	*null*	*null*	*null*	Gold	Física	87000
45565	CS-101	1	Primavera	2010	Katz	Informática	75000
45565	CS-319	1	Primavera	2010	Katz	Informática	75000
58583	*null*	*null*	*null*	*null*	Califieri	Historia	62000
76543	*null*	*null*	*null*	*null*	Singh	Finanzas	80000
76766	BIO-101	1	Verano	2009	Crick	Biología	72000
76766	BIO-301	1	Verano	2010	Crick	Biología	72000
83821	CS-190	1	Primavera	2009	Brandt	Informática	92000
83821	CS-190	2	Primavera	2009	Brandt	Informática	92000
83821	CS-319	2	Primavera	2010	Brandt	Informática	92000
98345	EE-181	1	Primavera	2009	Kim	Electrónica	80000

Figura 6.18. Resultado de *enseña* ⟖ *profesor*.

6.1.4.2. Agregación

La segunda operación del álgebra relacional extendida es la operación de agregación $\mathcal{G}$, que permite el uso de funciones de agregación como el mínimo o el promedio, sobre conjuntos de valores.

Las **funciones de agregación** toman una colección de valores y devuelven como resultado un único valor. Por ejemplo, la función de agregación **sum** toma una colección de valores y devuelve su suma. Por tanto, la función **sum** aplicada a la colección:

$$\{1, 1, 3, 4, 4, 11\}$$

devuelve el valor 24. La función de agregación **avg** devuelve la media de los valores. Cuando se aplica al conjunto anterior, devuelve el valor cuatro. La función de agregación **count** devuelve el número de elementos de la colección, que sería seis en el caso anterior. Otras funciones de agregación habituales son **min** y **max,** que devuelven los valores mínimo y máximo de la colección; en el ejemplo anterior devuelven uno y once, respectivamente.

Las colecciones sobre las que operan las funciones de agregación pueden contener valores repetidos; el orden en el que aparezcan los valores no tiene importancia. Estos conjuntos se denominan **multiconjuntos.** Los conjuntos son un caso especial de los multiconjuntos, en los que solo hay una copia de cada elemento.

Para ilustrar el concepto de agregación se usará la relación *profesor*. Suponga que se desea conocer la suma de todos los sueldos de los profesores; la expresión del álgebra relacional para esta consulta es la siguiente:

$$\mathcal{G}_{\mathbf{sum}(sueldo)}\,(profesor)$$

El símbolo $\mathcal{G}$ es la letra G en el tipo de letra caligráfico; se lee «G caligráfica». La operación del álgebra relacional $\mathcal{G}$ significa que se debe aplicar la agregación; y su subíndice especifica la operación

de agregación que se aplica. El resultado de la expresión anterior es una relación con un único atributo, que contiene una sola fila con un valor numérico correspondiente a la suma de todos los sueldos de todos los profesores.

Existen casos en los que se deben eliminar las ocurrencias repetidas de un valor antes de calcular una función de agregación. Si se desea eliminar los duplicados, se utiliza la misma función que antes, añadiendo con un guión la palabra «**distinct**» al final del nombre de la función (por ejemplo, **count-distinct**). Un ejemplo de este caso es «Encontrar el número total de profesores que enseñan una asignatura en el semestre de la primavera de 2010». En este caso cada profesor se cuenta solo una vez, independientemente del número de asignaturas que enseñe. La información requerida se encuentra en la relación *enseña*, por lo que se puede escribir la siguiente consulta:

$$\mathcal{G}_{\textbf{count-distinct}(ID)}((\sigma_{semestre\ =\ \text{«Primavera»}\ \wedge\ año\ =\ 2010}(enseña))$$

La función de agregación **count-distinct** asegura que aunque un profesor enseñe más de una asignatura, solo se cuenta una vez en el resultado.

Existen ocasiones en las que se desea aplicar las funciones de agregación no a un único conjunto de tuplas, sino a grupos de conjuntos de tuplas. Como ejemplo, suponga la siguiente consulta «Encontrar el sueldo medio de los distintos departamentos». Se escribe de la siguiente forma:

$$_{nombre_dept}\mathcal{G}_{\textbf{average}(sueldo)}(profesor)$$

La Figura 6.19 muestra las tuplas de la relación *profesor* agrupadas por el atributo *nombre_dept*. Este es el primer paso para el cálculo del resultado de la consulta. La función agregada indicada se calcula para cada uno de los grupos, y el resultado de las consultas se muestra en la Figura 6.20.

En contraste, considere la siguiente consulta «Calcular el sueldo medio de todos los profesores». Esta consulta se escribe de la siguiente forma:

$$\mathcal{G}_{\textbf{average}(sueldo)}(profesor)$$

En este caso, el atributo *nombre_dept* se ha omitido de la parte izquierda del operador $\mathcal{G}$, por lo que toda la relación se trata como un único grupo.

La forma general de la **operación de agregación** $\mathcal{G}$ es la siguiente:

$$_{G_1,\ G_2,\cdots\ G_n}\mathcal{G}_{F_1\ (A_1),\ F_2\ (A_2),\cdots\ F_m\ (A_m)}(E)$$

donde E es cualquier expresión del álgebra relacional; G_1, G_2,… G_n constituye una lista de atributos sobre los que se realiza la agrupación; cada F_i es una función de agregación y cada A_i es un nombre de atributo. El significado de la operación es el siguiente: las tuplas del resultado de la expresión E se dividen en grupos, de forma que:

1. Todas las tuplas de un grupo tienen los mismos valores para G_1, G_2,… G_n.
2. Las tuplas de diferentes grupos tiene valores diferentes para G_1, G_2,… G_n.

Por tanto, los grupos se pueden identificar por los valores de los atributos G_1, G_2,… G_n. Para cada grupo (g_1, g_2,… g_n), el resultado tiene una tupla (g_1, g_2,… g_n, a_1, a_2,… a_m) donde, para cada i, a_i es el resultado de aplicar la función de agregación F_i sobre el multiconjunto de valores para el atributo A_i en el grupo.

Como caso especial de la operación de agregación, la lista de los atributos G_1, G_2,… G_n puede estar vacía, en cuyo caso existe un único grupo que contiene todas las tuplas de la relación. Esto corresponde a la agregación sin agrupamiento.

ID	*nombre*	*nombre_dept*	*sueldo*
76766	Crick	Biología	72000
45565	Katz	Informática	75000
10101	Srinivasan	Informática	65000
83821	Brandt	Informática	92000
98345	Kim	Electrónica	80000
12121	Wu	Finanzas	90000
76543	Singh	Finanzas	80000
32343	El Said	Historia	60000
58583	Califieri	Historia	62000
15151	Mozart	Música	40000
33456	Gold	Física	87000
22222	Einstein	Física	95000

Figura 6.19. Tuplas de la relación *profesor*, agrupadas por el atributo *nombre_dept*.

nombre_dept	*sueldo*
Biología	72000
Informática	77333
Electrónica	80000
Finanzas	85000
Historia	61000
Música	40000
Física	91000

Figura 6.20. Relación resultado de la consulta «Calcular el sueldo medio de cada departamento».

ÁLGEBRA RELACIONAL DE MULTICONJUNTOS

Al contrario que en el álgebra relacional, SQL permite varias copias de una tupla con entrada de una relación, así como resultado de una consulta. La norma de SQL define cuántas copias de cada tupla se generan como resultado de una consulta, dependiendo de cuántas copias de tuplas existan en las relaciones de entrada.

Para modelar este comportamiento de SQL, se define una versión del álgebra relacional, denominada **álgebra relacional de multiconjuntos,** que opera sobre multiconjuntos, es decir, conjuntos que contienen duplicados. Las operaciones básicas del álgebra relacional de multiconjuntos se definen de la siguiente forma:

1. Si existen c_1 copias de la tupla t_1 en r_1, y t_1 satisface la selección σ_θ, entonces existen c_1 copias de t_1 en $\sigma\theta\ (r_1)$.
2. Para cada copia de la tupla t_1 en r_1, existe una copia de la tupla $\Pi_A(t_1)$ en $\Pi_A(r_1)$, donde $\Pi_A(t_1)$ indica la proyección de la única tupla t_1.
3. Si existen c_1 copias de la tupla t_1 en r_1 y c_2 copias de la tupla t_2 en r_2, existen $c_1 * c_2$ copias de la tupla $t_1.t_2$ en $r_1 \times r_2$.

Por ejemplo, suponga que las relaciones r_1 con el esquema (A, B) y r_2 con el esquema (C) son los siguientes multiconjuntos:

$$r_1 = \{(1, a), (2, a)\} \qquad r_2 = \{(2), (3), (3)\}$$

Entonces, $\Pi_B(r_1)$ sería $\{(a), (a)\}$, mientras que $\Pi_B(r_1) \times r_2$ sería:

$$\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$$

La unión, intersección y diferencia de multiconjuntos se pueden definir de forma similar, siguiendo las correspondientes definiciones en SQL, que se trataron en la Sección 3.5. No existe ninguna diferencia en la operación de agregación.

SQL Y EL ÁLGEBRA RELACIONAL

Para realizar una comparación de las operaciones del álgebra relacional y las de SQL, debería quedar claro que existe una fuerte conexión entre ambas. Una consulta típica de SQL tiene la forma:

select $A_1, A_2, \ldots A_n$
from $r_1, r_2, \ldots r_m$
where P

Cada A_i representa un atributo, y cada r_i una relación. P es un predicado. La consulta es equivalente a la expresión del álgebra relacional multiconjunto:

$$\Pi_{A_1, A_2, \ldots A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$$

Si se omite la cláusula **where,** el predicado P es **true.**

Se pueden escribir consultas más complejas en SQL que también se pueden escribir en el álgebra relacional. Por ejemplo la consulta:

select A_1, A_2, **sum**(A_3)
from $r_1, r_2, \ldots r_m$
where P
group by A_1, A_2

es equivalente a:

$$_{A_1, A_2}\mathcal{G}_{\text{sum}(A_3)} (\Pi_{A_1, A_2, \ldots A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m)))$$

Las expresiones de reunión de la cláusula **from** se pueden escribir en el álgebra relacional usando las expresiones de reunión equivalentes; se dejan los detalles como ejercicio para el lector. Sin embargo, las subconsultas de las cláusulas **where** y **select** no se pueden reescribir en álgebra relacional de forma tan directa, ya que no existe ninguna operación equivalente a la construcción subconsulta. Se han propuesto extensiones del álgebra relacional para esta tarea, pero quedan fuera del objeto del libro.

6.2. El cálculo relacional de tuplas

Cuando se escribe una expresión del álgebra relacional se proporcionan una serie de procedimientos que generan la respuesta a la consulta. El cálculo relacional de tuplas, en cambio, es un lenguaje de consultas **no procedimental.** Describe la información deseada sin establecer un procedimiento concreto para obtenerla.

Las consultas se expresan en el cálculo relacional de tuplas como:

$$\{t \mid P(t)\}$$

es decir, es el conjunto de todas las tuplas t tales que el predicado P es cierto para t. Siguiendo la notación usada anteriormente, se usa $t[A]$ para indicar el valor de la tupla t para el atributo A y $t \in r$ para indicar que la tupla t pertenece a la relación r.

Antes de dar una definición formal del cálculo relacional de tuplas se volverán a examinar algunas de las consultas para las que se escribieron expresiones del álgebra relacional en la Sección 6.1.1.

6.2.1. Ejemplos de consultas

Encontrar el *ID*, *nombre*, *nombre_dept* y *sueldo* de los profesores cuyo sueldo sea superior a 80.000 €:

$$\{t \mid t \in profesor \wedge t[sueldo] > 80000\}$$

Suponga que solo se desea obtener el atributo *ID*, en vez de todos los atributos de la relación *profesor*. Para escribir esta consulta en cálculo relacional de tuplas hay que incluir una expresión de relación sobre el esquema (*ID*). Se necesitan las tuplas de *ID* tales que hay una tupla de *profesor* con el atributo *sueldo* > 80.000. Para expresar esta consulta hay que usar la construcción «existe» de la lógica matemática. La notación:

$$\exists\, t \in r\, (Q(t))$$

significa «existe una tupla t de la relación r tal que el predicado $Q(t)$ es cierto».

Usando esta notación se puede escribir la consulta «Encontrar el *ID* de los profesores cuyo sueldo sea superior a 80.000 €», como:

$$\{t \mid \exists\, s \in profesor\, (t[ID] = s[ID] \wedge s[sueldo] > 80000)\}$$

En español la expresión anterior se lee «el conjunto de todas las tuplas t tales que existe una tupla s de la relación *profesor* para la que los valores de t y de s para el atributo *ID* son iguales y el valor de s para el atributo *sueldo* sea superior a 80.000 €».

La variable tupla t solo se define para el atributo *ID*, dado que es el único atributo para el que se especifica una condición para t. Por tanto, el resultado es una relación de *ID*.

Considere la consulta «Encontrar el nombre de todos los profesores cuyo departamento está en el edificio Watson». Esta consulta es un poco más compleja que las anteriores porque implica a dos relaciones: *profesor* y *departamento*. Como se verá, sin embargo, todo lo que necesita es que tengamos dos cláusulas «existe» en la expresión del cálculo relacional de tuplas, relacionadas mediante *y* ($\wedge$). La consulta se escribe de la siguiente manera:

$$\begin{aligned}&\{t \mid \exists\, s \in profesor\, (t[nombre] = s[nombre]\\ &\quad \wedge \exists\, u \in departamento\, (u[nombre_dept] = s[nombre_dept]\\ &\qquad\qquad \wedge\, u[edificio] = \text{«Watson»}))\}\end{aligned}$$

La variable u se restringe a los departamentos que están en el edificio Watson, mientras que la variable s se restringe a los profesores cuyo *nombre_dept* coincide con los de la variable u. En la Figura 6.21 se muestra el resultado de esta consulta.

Para encontrar el conjunto de todas las asignaturas que se enseñaron en el semestre del otoño de 2009, el semestre de la primavera de 2010 o en ambos, se utiliza la operación de unión del álgebra relacional. En el cálculo de tuplas se necesita añadir dos cláusulas «existe», conectadas con un *o* ($\vee$):

$$\begin{aligned}&\{t \mid \exists\, s \in sección\, (t[asignatura_id] = s[asignatura_id])\\ &\quad \wedge\, s[semestre] = \text{«Otoño»} \wedge s[año] = 2009)\}\\ &\vee \exists\, u \in sección\, (u[asignatura_id] = t[asignatura_id])\}\\ &\quad \wedge\, u[semestre] = \text{«Primavera»} \wedge u[año] = 2010)\}\end{aligned}$$

Esta expresión devuelve el conjunto de todas las tuplas con *asignatura_id* para las que se cumple al menos que:

- *asignatura_id* aparece en alguna tupla de la relación *sección* con *semestre* = Otoño y *año* = 2009.
- *asignatura_id* aparece en alguna tupla de la relación *sección* con *semestre* = Primavera y *año* = 2010.

nombre
Einstein
Crick
Gold

Figura 6.21. Nombre de todos los profesores cuyo departamento está en el edificio Watson.

Si la misma asignatura se ofrece tanto en los semestres del otoño de 2009 como en la primavera de 2010, su *asignatura_id* aparece solo en el resultado, ya que la definición matemática de conjunto no permite duplicados. El resultado de esta consulta se presentó anteriormente en la Figura 6.5.

Si *solo* queremos conocer los valores de *asignatura_id* de las asignaturas que se ofrecen tanto en el semestre del otoño de 2009, en el semestre de la primavera de 2010 o en ambos, todo lo que hay que hacer es cambiar en la expresión anterior la *o* ($\vee$) por una *y* ($\wedge$).

$$\begin{aligned}&\{t \mid \exists\, s \in \textit{sección}\ (t[\textit{asignatura_id}] = s[\textit{asignatura_id}])\\ &\quad \wedge\, s[\textit{semestre}] = \text{«Otoño»} \wedge s[\textit{año}] = 2009)\}\\ &\wedge\, \exists\, u \in \textit{sección}\ (u[\textit{asignatura_id}] = t[\textit{asignatura_id}])\}\\ &\quad \wedge\, u[\textit{semestre}] = \text{«Primavera»} \wedge u[\textit{año}] = 2010)\}\end{aligned}$$

El resultado de esta consulta se mostró en la Figura 6.13.

Considere ahora la consulta «Encontrar todas las asignaturas enseñadas en el semestre del otoño de 2009 pero no en el semestre de la primavera de 2010». La expresión del cálculo relacional de tuplas para esta consulta es parecida a las que se acaban de ver, salvo en el uso del símbolo *no* ($\neg$):

$$\begin{aligned}&\{t \mid \exists\, s \in \textit{sección}\ (t[\textit{asignatura_id}] = s[\textit{asignatura_id}])\\ &\quad \wedge\, s[\textit{semestre}] = \text{«Otoño»} \wedge s[\textit{año}] = 2009)\}\\ &\wedge\, \neg\, \exists\, u \in \textit{sección}\ (u[\textit{asignatura_id}] = t[\textit{asignatura_id}])\}\\ &\quad \wedge\, u[\textit{semestre}] = \text{«Primavera»} \wedge u[\textit{año}] = 2010)\}\end{aligned}$$

Esta expresión del cálculo relacional de tuplas usa la cláusula $\exists s \in$ *sección* (…) para exigir que la *asignatura* se enseñe en el semestre del otoño de 2009, y la $\neg\, \exists\, u \in$ *sección* (…) para eliminar aquellos valores de *asignatura_id* que aparecen en alguna tupla de la relación *sección* por haberse enseñado en el semestre de la primavera de 2010.

La consulta que se tomará ahora en consideración usa la implicación, que se indica como $\rightarrow$. La fórmula $P \rightarrow Q$ significa «*P* implica *Q*»; es decir, «si *P* es cierto, entonces *Q* tiene que serlo también». Observe que $P \rightarrow Q$ es equivalente lógicamente a $\neg P \vee Q$. El empleo de la implicación en lugar de *no* y *o* sugiere una interpretación más intuitiva de la consulta en español.

Considere la consulta «Encontrar todos los clientes que se hayan matriculado en asignaturas que oferta el departamento de Biología». Para escribir esta consulta en el cálculo relacional de tuplas se introduce la construcción «para todo», que se indica como $\forall$. La notación:

$$\forall\, t \in r\ (Q(t))$$

significa «*Q* es verdadera para todas las tuplas *t* de la relación *r*».

La expresión para la consulta se escribe de la siguiente manera:

$$\begin{aligned}&\{t \mid \exists\, r \in \textit{estudiante}\ (r\,[ID] = t[ID]) \wedge\\ &\quad (\forall\, u \in \textit{asignatura}\ (u[\textit{nombre_dept}] = \text{«Biología»} \rightarrow\\ &\qquad \exists\, s \in \textit{matricula}\ (t[ID] = s[ID]\\ &\qquad \wedge\, s[\textit{asignatura_id}] = u[\textit{asignatura_id}]))\}\end{aligned}$$

En español, esta expresión se interpreta como «el conjunto de todos los estudiantes (es decir, las tuplas *t* (*ID*)) tales que, para *todas* las tuplas *u* de la relación *asignatura*, si el valor de *u* en el atributo *nombre_dept* es 'Biología', entonces existe una tupla en la relación *matricula* que incluye al estudiante *ID* y *asignatura_id*».

Observe que hay cierta sutileza en la consulta anterior: si no hay ninguna asignatura que se oferte en el departamento de Biología, todos los estudiantes satisfacen la condición. La primera línea de la expresión de consulta es crítica en este caso; sin la condición:

$$\exists\, r \in \textit{estudiante}\ (r[ID] = t[ID])$$

si no hay ninguna asignatura que se oferte en el departamento de Biología, cualquier valor de *t* (incluyendo los que no son *ID* de estudiantes de la relación *estudiante*) valdría.

6.2.2. Definición formal

Ahora ya está preparado para una definición formal. Las expresiones del cálculo relacional de tuplas son de la forma:

$$\{t \mid P\,(t)\}$$

donde *P* es una *fórmula*. En una fórmula pueden aparecer varias variables tupla. Se dice que una variable tupla es una *variable libre* a menos que esté cuantificada mediante $\exists$ o $\forall$. Por tanto, en:

$$\begin{aligned}&t \in \textit{profesor} \wedge \exists\, s \in \textit{departamento}\\ &\qquad (t[\textit{nombre_dept}] = s[\textit{nombre_dept}])\end{aligned}$$

t es una variable libre. La variable tupla *s* se denomina variable *ligada*.

Las fórmulas del cálculo relacional de tuplas se construyen con *átomos*. Los átomos tienen una de las siguientes formas:

- $s \in r$, donde *s* es una variable tupla y *r* es una relación (no se permite el uso del operador $\notin$)
- $s[x]\ \Theta\ u[y]$, donde *s* y *u* son variables tuplas, *x* es un atributo en el que está definida *s*, *y* es un atributo en el que está definida *u*, y Θ es un operador de comparación ($<, \leq, =, \neq, >, \geq$); se exige que los atributos *x* e *y* tengan dominios cuyos miembros puedan compararse mediante Θ.
- $s[x]\ \Theta\ c$, donde *s* es una variable tupla, *x* es un atributo en el que está definida *s*, Θ es un operador de comparación y *c* es una constante en el dominio del atributo *x*.

Las fórmulas se construyen a partir de los átomos usando las siguientes reglas:

- Un átomo es una fórmula.
- Si P_1 es una fórmula, también lo son $\neg P_1$ y (P_1).
- Si P_1 y P_2 son fórmulas, también lo son $P_1 \vee P_2$, $P_1 \wedge P_2$ y $P_1 \rightarrow P_2$.
- Si $P_1(s)$ es una fórmula que contiene la variable tupla libre *s*, y *r* es una relación, entonces:

$$\exists\, s \in r\ (P_1\,(s)) \text{ y } \forall\, s \in r\ (P_1\,(s))$$

también son fórmulas.

Igual que en el álgebra relacional, se pueden escribir expresiones equivalentes que no sean idénticas en apariencia. En el cálculo relacional de tuplas estas equivalencias incluyen las tres reglas siguientes:

1. $P_1 \wedge P_2$ es equivalente a $\neg(\neg(P_1) \vee \neg(P_2))$.
2. $\forall\, t \in r\ (P_1(\mathrm{t}))$ es equivalente a $\neg\, \exists\, t \in r\ (\neg P_1(t))$.
3. $P_1 \rightarrow P_2$ es equivalente a $\neg(P_1) \vee P_2$.

6.2.3. Seguridad de las expresiones

Queda un último asunto por tratar. Las expresiones del cálculo relacional de tuplas pueden generar relaciones infinitas. Suponga que se escribe la expresión:

$$\{t \mid \neg\ (t \in \textit{profesor})\}$$

Hay infinitas tuplas que no están en *profesor*. La mayor parte de estas tuplas contienen valores que ¡ni siquiera aparecen en la base de datos! Resulta evidente que no se desea permitir expresiones de ese tipo.

Para ayudar a definir las restricciones del cálculo relacional de tuplas se introduce el concepto de **dominio** de las fórmulas relacionales de tuplas, *P*. De manera intuitiva, el dominio de *P*, que se indica como *dom*(*P*), es el conjunto de todos los valores a los que *P* hace referencia. Esto incluye a los valores mencionados en la propia *P*, así como a los valores que aparezcan en tuplas de relaciones mencionadas por *P*. Por tanto, el dominio de *P* es el conjunto de todos los valores que aparecen explícitamente en *P*, o en una o más relaciones cuyos nombres aparezcan en *P*. Por ejemplo, $dom(t \in \textit{profesor} \wedge \mathrm{t}[\textit{sueldo}] > 80.000)$ es el conjunto que contiene a 80.000 y

el conjunto de todos los valores que aparecen en cualquier atributo de cualquier tupla de la relación *profesor*. Además, $dom(\neg\ (t \in profesor))$ es el conjunto de todos los valores que aparecen en *profesor*, dado que la relación *profesor* se menciona en la expresión.

Se dice que una expresión $\{t \mid P(t)\}$ es *segura* si todos los valores que aparecen en el resultado son valores de $dom(P)$. La expresión $\{t \mid (t \in profesor)\}$ no es segura. Observe que $dom(\neg\ (t \in profesor))$ es el conjunto de todos los valores que aparecen en *profesor*. Sin embargo, es posible tener una tupla t que no esté en *profesor* que contenga valores que no aparezcan en *profesor*. El resto de ejemplos de expresiones del cálculo relacional de tuplas que se han escrito en esta sección son seguros.

El número de tuplas que satisface una expresión no segura, como $\{t \mid \neg(t \in profesor)\}$, podría ser infinita, mientras que para las expresiones seguras se tiene la garantía de que sus resultados son finitos. La clase de expresiones del cálculo relacional de tuplas que se permiten se restringe, por tanto, a las que son seguras.

6.2.4. Potencia expresiva de los lenguajes

El cálculo relacional de dominios restringido a las expresiones seguras es equivalente en potencia expresiva al álgebra relacional básico (con los operadores $\cup$, $-$, $\times$, σ, y ρ, pero sin las operaciones relacionales extendidas, como la proyección generalizada y la agregación ($\mathcal{G}$)). Por tanto, todas las expresiones del álgebra relacional solo utilizan las operaciones básicas, existe una expresión equivalente en el cálculo relacional de tuplas y todas las expresiones del cálculo relacional de tuplas tienen una expresión equivalente en el álgebra relacional. No se va a demostrar en este momento; en las notas bibliográficas puede encontrar referencias a la demostración. Partes de la demostración se incluyen en los ejercicios. Hay que indicar que el cálculo relacional de tuplas no tiene una operación equivalente a la agregación, pero se puede extender para incorporarla. La extensión del cálculo relacional de tuplas para manejar expresiones aritméticas es directa.

6.3. El cálculo relacional de dominios

Una segunda forma de cálculo relacional, denominada **cálculo relacional de dominios,** usa variables de dominio, que toman sus valores del dominio de un atributo, en vez de hacerlo para una tupla completa. El cálculo relacional de dominios, no obstante, se halla estrechamente relacionado con el cálculo relacional de tuplas.

El cálculo relacional de dominios sirve de base teórica al ampliamente utilizado lenguaje QBE (véase Apéndice C.1 en red), al igual que el álgebra relacional sirve como base para el lenguaje SQL.

6.3.1. Definición formal

Las expresiones del cálculo relacional de dominios son de la forma:

$$\{<x_1, x_2, \ldots x_n> \mid P\,(x_1, x_2, \ldots x_n)\}$$

donde $x_1, x_2, \ldots x_n$ representan las variables de dominio. P representa una fórmula compuesta por átomos, como era el caso en el cálculo relacional de tuplas. Los átomos del cálculo relacional de dominios tienen una de las formas siguientes:

- $<x_1, x_2, \ldots x_n> \in r$, donde r es una relación con n atributos y x_1, $x_2, \ldots x_n$ son variables de dominio o constantes de dominio.
- $x\ \Theta\ y$, donde x e y son variables de dominio y Θ es un operador de comparación ($<, \leq, =, \neq, >, \geq$). Se exige que los atributos x e y tengan dominios que puedan compararse mediante Θ.
- $x\ \Theta\ c$, donde x es una variable de dominio, Θ es un operador de comparación y c es una constante del dominio del atributo para el que x es una variable de dominio.

Las fórmulas se construyen a partir de los átomos usando las reglas siguientes:

- Los átomos son fórmulas.
- Si P_1 es una fórmula, también lo son P_1 y (P_1).
- Si P_1 y P_2 son fórmulas, también lo son $P_1 \vee P_2$, $P_1 \wedge P_2$ y $P_1 \rightarrow P_2$.
- Si $P_1\ (x)$ es una fórmula en x, donde x es una variable libre de dominio, entonces:

$$\exists\, x\ (P_1\ (x)) \text{ y } \forall\, x\ (P_1\ (x))$$

también son fórmulas.

Como notación abreviada se escribe $\exists\, a, b, c\ (P\ (a, b, c))$ en lugar de $\exists\, a\ (\exists\, b\ (\exists\, c\ (P\ (a, b, c))))$.

6.3.2. Ejemplos de consultas

Ahora se van a presentar consultas del cálculo relacional de dominios para los ejemplos considerados anteriormente. Observe la similitud de estas expresiones con las expresiones correspondientes del cálculo relacional de tuplas.

- Encontrar el *ID*, *nombre*, *nombre_dept* y *sueldo* de los profesores cuyo sueldo sea superior a 80.000 €:

$$\{<i, n, d, s> \mid <i, n, d, s> \in profesor \wedge s > 80000\}$$

- Encontrar todos los *ID* de profesor para los profesores cuyo sueldo sea superior a 80.000 €:

$$\{<n> \mid \exists\, i, d\,, s\ (<i, n, d, s> \in profesor \wedge > 80000)\}$$

Aunque la segunda consulta tenga un aspecto muy parecido al de la consulta que se escribió para el cálculo relacional de tuplas, hay una diferencia importante. En el cálculo de tuplas, cuando se escribe $\exists\, s$ para alguna variable tupla s, se vincula inmediatamente con una relación escribiendo $\exists\, s \in r$. Sin embargo, cuando se usa $\exists\, n$ en el cálculo de dominios, n no se refiere a una tupla, sino a un valor del dominio. Por tanto, el dominio de la variable n no está restringido hasta que la subfórmula $<i, n, d, s> \in profesor$ restringe n a los nombres de profesores que aparecen en la relación *profesor*.

Ahora se mostrarán varios ejemplos de consultas del cálculo relacional de dominios.

- Encontrar los nombres de todos los profesores del departamento de Física junto con los *asignatura_id* de todas las asignaturas que enseñan:

$$\begin{aligned}\{<n, c> \mid \exists\, i, a\ (&<i, c, a, s, y> \in enseña \\ &\wedge \exists\, d, s\ (<i, n, d, s> \in profesor \wedge d = \text{«Física»}))\}\end{aligned}$$

- Encontrar el conjunto de todas las asignaturas que se enseñaron en el semestre del otoño de 2009, en el semestre de la primavera de 2010 o en ambos:

$$\begin{aligned}\{<c> \mid \exists\, s\ (&<c, a, s, y, b, r, t> \in sección \\ &\wedge s = \text{«Otoño»} \wedge y = \text{«2009»} \\ \vee\ \exists\, u\ (&<c, a, s, y, b, r, t> \in sección \\ &\wedge s = \text{«Primavera»} \wedge y = \text{«2010»}\end{aligned}$$

- Encontrar todos los estudiantes que se han matriculado en asignaturas que ofrece el departamento de Biología:

$$\begin{aligned}\{<i> \mid \exists\, n, d, t\ (&<i, n, d, t> \in estudiante) \wedge \\ &\forall\, x, y, z, w\ (<x, y, z, w> \in asignatura \wedge \\ &z = \text{«Biología»} \rightarrow \\ &\exists\, a, b\ (<a, x, b, r, p, q> \in matricula \wedge \\ &<c, a> \in departamento))\}\end{aligned}$$

Observe que como en el caso del cálculo relacional de tuplas, si el departamento de Biología no ofertó ninguna asignatura, en el resultado estarían todos los estudiantes.

6.3.3. Seguridad de las expresiones

Ya se observó que, en el cálculo relacional de tuplas (Sección 6.2), es posible escribir expresiones que generen relaciones infinitas. Esto llevó a definir la *seguridad* de las expresiones de cálculo relacional de tuplas. En el cálculo relacional de dominios se produce una situación parecida. Las expresiones como:

$$\{<i, n, d, s> \mid \neg(<i, n, d, s> \in profesor)\}$$

no son seguras porque permiten valores del resultado que no están en el dominio de la expresión.

En el cálculo relacional de dominios también hay que tener en cuenta la forma de las fórmulas dentro de las cláusulas «existe» y «para todo». Considere la expresión:

$$\{<x> \mid \exists\, y\, (<x, y> \in r) \wedge \exists\, z\, (\neg(<x, z> \in r) \wedge P\,(x, z))\}$$

donde P es una fórmula que implica a x y a z. Se puede comprobar la primera parte de la fórmula, $\exists\, y\, (<x, y> \in r)$, tomando en consideración solo los valores de r. Sin embargo, para comprobar la segunda parte de la fórmula, $\exists\, z\, (\neg(<x, z> \in r) \wedge P\,(x, z))$, hay que tomar en consideración valores de z que no aparecen en r. Dado que todas las relaciones son finitas, no aparece en r un número infinito de valores. Por tanto, en general no resulta posible comprobar la segunda parte de la fórmula sin tomar en consideración un número infinito de valores posibles de z. En su lugar se añaden restricciones para prohibir expresiones como la anterior.

En el cálculo relacional de tuplas, se restringió a una determinada relación el alcance posible de cualquier variable cuantificada. Dado que no se hizo así en el cálculo de dominios, se añaden reglas a la definición de seguridad para tratar casos como el del ejemplo. Se dice que la expresión:

$$\{<x_1, x_2, \ldots x_n> \mid P\,(x_1, x_2, \ldots x_n)\}$$

es segura si se cumplen todas las condiciones siguientes:

1. Todos los valores que aparecen en las tuplas de la expresión son valores de $dom(P)$.
2. Para cada subfórmula «existe» de la forma $\exists\, x\, (P_1\,(x))$ la subfórmula es cierta si y solo si hay un valor x de $dom(P_1)$ tal que $P_1(x)$ es cierto.
3. Para cada subfórmula «para todo» de la forma $\forall\, x\, (P_1(x))$ la subfórmula es verdadera si y solo si $P_1\,(x)$ es cierto para todos los valores x de $dom(P_1)$.

El propósito de las reglas adicionales es garantizar que se puedan comprobar las subfórmulas «para todo» y «existe» sin tener que comprobar infinitas posibilidades.

Considere la segunda regla de la definición de seguridad. Para que $\exists\, x\, (P_1\,(x))$ sea cierto solo hay que encontrar una x para la que $P_1(x)$ lo sea.

En general habría que comprobar infinitos valores. Sin embargo, si la expresión es segura, se sabe que se puede restringir la atención a los valores de $dom(P_1)$. Esta restricción reduce las tuplas que hay que tomar en consideración a un número finito.

La situación de las subfórmulas de la forma $\forall x\, (P_1\,(x))$ es parecida. Para asegurar que $\forall x\, (P_1\,(x))$ es cierto, en general hay que comprobar todos los valores posibles, por lo que es necesario examinar infinitos valores. Como antes, si se sabe que la expresión es segura, basta con comprobar $P_1\,(x)$ para los valores tomados de $dom(P_1)$.

Todas las expresiones del cálculo relacional de dominios que se han incluido en los ejemplos de consultas de esta sección son seguras, excepto la del ejemplo de consulta no segura que se vio anteriormente.

6.3.4. Potencia expresiva de los lenguajes

Cuando el cálculo relacional de dominios se restringe a las expresiones seguras, es equivalente en potencia expresiva al cálculo relacional de tuplas restringido también a las expresiones seguras.

Dado que ya se observó anteriormente que el cálculo relacional de tuplas restringido es equivalente al álgebra relacional, los tres lenguajes siguientes son equivalentes:

- El álgebra relacional básica (sin las operaciones extendidas del álgebra relacional).
- El cálculo relacional de tuplas restringido a las expresiones seguras.
- El cálculo relacional de dominios restringido a las expresiones seguras.

Hay que tener en cuenta que el cálculo relacional de dominios tampoco tiene equivalente para la operación agregación, pero se puede extender fácilmente para contenerla, y extenderlo para manejar expresiones aritméticas es inmediato.

6.4. Resumen

- El **álgebra relacional** define un conjunto de operaciones algebraicas que operan sobre las tablas y devuelven tablas como resultado. Estas operaciones se pueden combinar para obtener expresiones que expresen las consultas deseadas. El álgebra define las operaciones básicas usadas en los lenguajes de consultas relacionales.
- Las operaciones del álgebra relacional se pueden dividir en:
  - Operaciones básicas.
  - Operaciones adicionales que se pueden expresar en términos de las operaciones básicas.
  - Operaciones extendidas, algunas de las cuales añaden mayor poder expresivo al álgebra relacional.
- El álgebra relacional es un lenguaje rígido y formal que no resulta adecuado para los usuarios ocasionales de los sistemas de bases de datos. Los sistemas comerciales de bases de datos, por tanto, usan lenguajes con más «azúcar sintáctico». En los Capítulos 3 al 5 se trata el lenguaje más influyente, SQL, que está basado en el álgebra relacional.
- El **cálculo relacional de tuplas** y el **cálculo relacional de dominios** son lenguajes no procedimentales que representan la potencia básica necesaria en un lenguaje de consultas relacionales. El álgebra relacional básica es un lenguaje procedimental que es equivalente en potencia a ambas formas del cálculo relacional cuando se limitan a expresiones seguras.
- Los cálculos relacionales son lenguajes rígidos y formales que no resultan adecuados para los usuarios ocasionales de los sistemas de bases de datos. Estos dos lenguajes formales son la base de dos lenguajes más amigables, QBE y Datalog, que se tratan en el Apéndice C (véase en red).

Términos de repaso

- Álgebra relacional.
- Operaciones del álgebra relacional.
  - Selección (σ).
  - Proyección (Π).
  - Unión ($\cup$).
  - Diferencia de conjuntos ($-$).
  - Producto cartesiano ($\times$).
  - Renombramiento (ρ).
- Operaciones adicionales.
  - Intersección de conjuntos ($\cap$).
  - Reunión natural $\bowtie$.
  - Operación asignación.
  - Reunión externa.
    - Reunión externa por la izquierda $⟕$.
    - Reunión externa por la derecha $⟖$.
    - Reunión externa completa $⟗$.
- Multiconjuntos.
- Agrupación.
- Valores nulos.
- Cálculo relacional de tuplas.
- Cálculo relacional de dominios.
- Seguridad de las expresiones.
- Potencia expresiva de los lenguajes.

Ejercicios prácticos

6.1. Escriba las siguientes consultas en álgebra relacional, usando el esquema de la universidad:

a. Encontrar los nombres de las asignaturas del departamento de Informática que tienen tres créditos.

b. Encontrar los ID de todos los estudiantes que tuvieron clase con un profesor llamado Einstein; asegúrese de que no haya duplicados en el resultado.

c. Encontrar el mayor sueldo de todos los profesores.

d. Encontrar todos los profesores que ganen el mayor sueldo (puede haber más de uno con el mismo sueldo).

e. Encontrar la matrícula de cada sección que se ofertó en el otoño de 2009.

f. Encontrar la mayor matrícula, entre todas las secciones, en el otoño de 2009.

g. Encontrar las secciones que tuvieron la mayor matrícula en el otoño de 2009.

6.2. Considere la base de datos relacional de la Figura 6.22, en la que las claves primarias están subrayadas. Obtenga una expresión del álgebra relacional para cada una de las siguientes consultas:

a. Encontrar el nombre de todos los empleados que viven en la misma ciudad y en la misma calle que sus jefes.

b. Encontrar el nombre de todos los empleados de esta base de datos que no trabajan para «First Bank Corporation».

c. Encontrar el nombre de todos los empleados que ganan más que cualquier empleado de «Small Bank Corporation».

6.3. Las operaciones de reunión externa extienden la operación reunión natural de manera que las tuplas de las relaciones participantes no se pierdan en el resultado de la reunión. Describa la manera en que la operación reunión zeta puede extenderse para que las tuplas de la relación de la izquierda, las de la relación de la derecha o las de ambas relaciones no se pierdan en el resultado de las reuniones zeta.

6.4. (Operación división). El operador división del álgebra relacional, «÷», se define como sigue. Sean r (R) y $s(S)$ relaciones, donde $S \subseteq R$; es decir, todos los atributos del esquema S también lo son del esquema R. Entonces, $r \div s$ es una relación sobre el esquema $R - S$ (es decir, sobre el esquema que contiene todos los atributos del esquema R que no están en el esquema S). Una tupla t está en $r \div s$ si y solo si se cumplen estas dos condiciones:

- t está en $\Pi_{R-S}(r)$
- Para toda tupla t_s en s, existe una tupla t_r en r que satisface las dos condiciones siguientes:
  a. $t_r\,[S] = t_s\,[S]$
  b. $t_r\,[R - S] = t$

Dadas las definiciones anteriores:

a. Escriba una expresión del álgebra relacional usando el operador división para encontrar los ID de todos los estudiantes que hayan estado matriculados en todas las asignaturas de Informática. (Sugerencia: proyecte *matricula* solo con los ID y *asignatura_id*, y genere el conjunto de todos los *asignatura_id* de Informática utilizando una expresión de selección, antes de realizar la división).

b. Indique cómo escribir la consulta anterior en álgebra relacional, sin utilizar la división. (De esta forma, podría demostrar cómo definir el operador de división utilizando el resto de operaciones del álgebra relacional).

6.5. Sean los siguientes esquemas de relaciones:

$R = (A, B, C)$

$S = (D, E, F)$

Sean las relaciones $r(R)$ y $s(S)$. Indique una expresión del cálculo relacional de tuplas que sea equivalente a cada una de las siguientes:

a. $\Pi_A(r)$

b. $\sigma_{B = 17}(r)$

c. $r \times s$

d. $\Pi_{A,\,F}(\sigma_{C = D}(r \times s))$

6.6. Sea $R = (A, B, C)$ y sean r_1 y r_2 relaciones del esquema R. Indique una expresión del cálculo relacional de dominios que sea equivalente a cada una de las siguientes:

a. $\Pi_A(r_1)$

b. $\sigma_{B = 17}(r_1)$

c. $r_1 \cup r_2$

d. $r_1 \cap r_2$

e. $r_1 - r_2$

f. $\Pi_{A,\,B}(r_1) \bowtie \Pi_{B,\,C}(r_2)$

6.7. Sea $R = (A, B)$ y $S = (A, C)$, y sean r (R) y $s(S)$ relaciones. Escriba expresiones del álgebra relacional para cada una de las siguientes consultas:

a. $\{<a> \mid \exists\, b\, (<a, b> \in r \wedge b = 7)\}$

b. $\{<a, b, c> \mid <a, b> \in r \wedge <a\, , c> \in s\}$

c. $\{<a> \mid \exists\, c\, (<a, c> \in s \wedge \exists\, b_1, b_2\, (<a, b_1> \in r \wedge <c, b_2> \in r \wedge b_1 > b_2))\}$

6.8. Considere la base de datos relacional de la Figura 6.22, en la que las claves primarias están subrayadas. Obtenga una expresión del cálculo relacional de tuplas para cada una de las siguientes consultas:

a. Encontrar todos los empleados que trabajan directamente para «Jones».

b. Encontrar todas las ciudades de residencia de todos los empleados que trabajan directamente para «Jones».

c. Encontrar el nombre del jefe de «Jones».

d. Encontrar a aquellos empleados que ganan más que todos los empleados que viven en la ciudad de «Mumbai».

6.9. Describa cómo traducir expresiones de reunión en SQL a álgebra relacional.

empleado(*nombre_persona*, *calle*, *ciudad*)
trabaja(*nombre_persona*, *nombre_empresa*, *sueldo*)
empresa(*nombre_empresa*, *ciudad*)
jefe(*nombre_persona*, *nombre_jefe*)

Figura 6.22. Base de datos relacional para los Ejercicios 6.2, 6.8, 6.11, 6.13 y 6.15.

Ejercicios

6.10. Escriba las siguientes consultas en álgebra relacional, usando el esquema de la universidad:

a. Encontrar los nombres de todos los estudiantes que hayan tenido al menos una asignatura de Informática.

b. Encontrar los ID y los nombres de todos los estudiantes que no se hayan matriculado en ninguna asignatura ofertada antes de la primavera de 2009.

c. Para cada departamento, encontrar el mayor sueldo de profesores de dicho departamento. Puede suponer que todos los departamentos tienen al menos un profesor.

d. Encontrar el menor sueldo, entre todos los departamentos, de los máximos sueldos por departamento calculado según la consulta anterior.

6.11. Considere la base de datos relacional de la Figura 6.22, en la que se han subrayado las claves primarias Proporcione expresiones del álgebra relacional para expresar cada una de las siguientes consultas:

a. Encontrar el nombre de todos los empleados que trabajan en «First Bank Corporation».

b. Encontrar el nombre y la ciudad de residencia de todos los empleados que trabajan en «First Bank Corporation».

c. Encontrar el nombre, la dirección y la ciudad de residencia de todos los empleados que trabajan en «First Bank Corporation» y ganan más de 10.000 €.

d. Encontrar el nombre de todos los empleados de esta base de datos que viven en la misma ciudad que la compañía para la que trabajan.

e. Suponga que las empresas pueden tener sede en varias ciudades. Encontrar todas las empresas con sede en todas las ciudades en las que tiene sede «Small Bank Corporation».

6.12. Usando el ejemplo de la universidad, escriba consultas del álgebra relacional para encontrar las secciones de asignaturas que enseñan más de un profesor de las siguientes formas:

a. Usando una función de agregación.

b. Sin usar funciones de agregación.

6.13. Considere la base de datos relacional de la Figura 6.22. Obtenga una expresión del álgebra relacional para cada una de las siguientes consultas:

a. Encontrar la compañía con mayor número de empleados.

b. Encontrar la compañía con la nómina más reducida.

c. Encontrar las compañías cuyos empleados ganen un sueldo más elevado, de media, que el sueldo medio en «First Bank Corporation».

6.14. Considere el siguiente esquema de relación para una biblioteca:

socio(*socio_núm*, *nombre*, *dob*)
libros(*isbn*, *título*, *autores*, *editor*)
préstamo(*socio_núm*, *isbn*, *fecha*)

Escriba las consultas siguientes en el álgebra relacional.

a. Encontrar el nombre de los socios que han tomado prestado algún libro editado por «McGraw-Hill».

b. Encontrar el nombre de los socios que han tomado prestados todos los libros editados por «McGraw-Hill».

c. Encontrar el nombre de los socios que han tomado prestados más de cinco libros diferentes editados por «McGraw-Hill».

d. Para cada editorial, encontrar el nombre y los números de los socios que han tomado prestados más de cinco libros de esa editorial.

e. Encontrar el número medio de libros que ha tomado en préstamo cada socio. Tenga en cuenta que si un socio no toma prestado ningún libro, entonces no aparece en la relación *préstamo*.

6.15. Considere la base de datos de empleados de la Figura 5.14. Proporcione expresiones del cálculo relacional de tuplas para cada una de las siguientes consultas:

a. Encontrar el nombre de todos los empleados que trabajan en «First Bank Corporation».

b. Encontrar el nombre y la ciudad de residencia de todos los empleados que trabajan en «First Bank Corporation».

c. Encontrar el nombre, la dirección y la ciudad de residencia de todos los empleados que trabajan en «First Bank Corporation» y ganan más de 10.000 €.

d. Encontrar el nombre de todos los empleados de esta base de datos que viven en la misma ciudad que la compañía para la que trabajan.

e. Encontrar todos los empleados que viven en la misma ciudad y en la misma calle que su jefe.

f. Encontrar todos los empleados de la base de datos que no trabajan en «First Bank Corporation».

g. Encontrar todos los empleados que ganan más que cualquier empleado de «Small Bank Corporation».

h. Suponga que las empresas pueden tener sede en varias ciudades. Encontrar todas las empresas con sede en todas las ciudades en las que tiene sede «Small Bank Corporation».

6.16. Sea $R = (A, B)$ y $S = (A, C)$, y sean r (R) y s(S) relaciones. Escriba expresiones del álgebra relacional equivalentes a cada una de las expresiones siguientes del cálculo relacional de dominios:

a. $\{<a> \mid \exists\, b\, (<a, b> \in r \wedge b = 17)\}$

b. $\{<a, b, c> \mid <a, b> \in r \wedge <a, c> \in s\}$

c. $\{<a> \mid \exists\, b\, (<a, b> \in r) \vee \forall\, c\, (\exists\, d\, (<d, c> \in s) \rightarrow <a, c> \in s)\}$

d. $\{<a> \mid \exists\, c\, (<a, c> \in s \wedge \exists\, b_1, b_2\, (<a, b_1> \in r \wedge <c, b_2> \in r \wedge b_1 > b_2))\}$

6.17. Repita el Ejercicio 6.16 escribiendo consultas SQL en lugar de expresiones del álgebra relacional.

6.18. Sea $R = (A, B)$ y $S = (A, C)$, y sean r (R) y $s(S)$ relaciones. Usando la constante especial *null*, escriba expresiones del cálculo relacional de tuplas equivalentes a cada una de las siguientes:

a. $r \bowtie s$

b. $r ⟕ s$

c. $r ⟖ s$

6.19. Indique una expresión del cálculo relacional de tuplas para calcular el valor máximo de r (A).

Notas bibliográficas

La definición original del álgebra relacional está en Codd [1970]. En Codd [1979] se presentan extensiones del modelo relacional y se describe la incorporación de los valores nulos al álgebra relacional (el modelo RM/T), así como la de las reuniones externas. Codd [1990] es un compendio de los trabajos de E.F. Codd sobre el modelo relacional. Las reuniones externas también se estudian en Date [1993b].

La definición original del cálculo relacional de tuplas aparece en Codd [1972]. Hay una prueba formal de la equivalencia del cálculo relacional de tuplas y del álgebra relacional en Codd [1972]. Se han propuesto varias extensiones del cálculo relacional de tuplas. Klug [1982] y Escobar-Molano et ál. [1993] describen extensiones para funciones de agregación escalares.

Parte 2
Diseño de bases de datos

Los sistemas de bases de datos están diseñados para gestionar grandes cantidades de información. Estas grandes cantidades de información no se encuentran aisladas. Forman parte del funcionamiento de alguna empresa cuyo producto final puede ser la información obtenida de la base de datos o algún producto o servicio para el que la base de datos solo desempeña un papel secundario.

Los dos primeros capítulos de esta parte se centran en el diseño de los esquemas de las bases de datos. El modelo entidad-relación (E-R) descrito en el Capítulo 7 es un modelo de datos de alto nivel. En lugar de representar todos los datos en tablas, distingue entre los objetos básicos, denominados *entidades*, y las *relaciones* entre esos objetos. Suele utilizarse como un primer paso en el diseño de los esquemas de las bases de datos.

El diseño de las bases de datos relacionales —el diseño del esquema relacional— se trató informalmente en los capítulos anteriores. No obstante, existen criterios para distinguir los buenos diseños de bases de datos de los malos. Estos se formalizan mediante varias «formas normales» que ofrecen diferentes compromisos entre la posibilidad de inconsistencias y la eficiencia de determinadas consultas. El Capítulo 8 describe el diseño formal de los esquemas de las relaciones.

El diseño de un entorno completo de aplicaciones para bases de datos, que responda a las necesidades de la empresa que se modela, exige prestar atención a un conjunto de aspectos más amplio, muchos de los cuales se tratan en el Capítulo 9. Este capítulo describe en primer lugar el diseño de las interfaces basadas en Web para aplicaciones. Finalmente el capítulo proporciona una detallada discusión sobre la seguridad en los niveles de aplicación y de base de datos.

Diseño de bases de datos y el modelo E-R

07

Hasta este punto se ha dado por supuesto un determinado esquema de la base de datos y se ha estudiado la manera de expresar las consultas y las actualizaciones. Ahora se va a considerar en primer lugar la manera de diseñar el esquema de la base de datos. En este capítulo nos centraremos en el modelo de datos entidad-relación (E-R), que ofrece una manera de identificar las entidades que se van a representar en la base de datos y el modo en que se relacionan entre sí. Finalmente, el diseño de la base de datos se expresará en términos del diseño de bases de datos relacionales y del conjunto de restricciones asociado. En este capítulo se mostrará la manera en que el diseño E-R puede transformarse en un conjunto de esquemas de relación y el modo en que se pueden incluir algunas de las restricciones en ese diseño. Luego, en el Capítulo 8, se considerará en detalle si el conjunto de esquemas de relación representa un diseño de la base de datos bueno o malo y se estudiará el proceso de creación de buenos diseños usando un conjunto de restricciones más amplio. Estos dos capítulos tratan los conceptos fundamentales del diseño de bases de datos.

7.1. Visión general del proceso de diseño

La tarea de creación de aplicaciones de bases de datos es una labor compleja, que implica varias fases, como el diseño del esquema de la base de datos, el diseño de los programas que tienen acceso a los datos y los actualizan, así como el diseño del esquema de seguridad para controlar el acceso a los datos. Las necesidades de los usuarios desempeñan un papel central en el proceso de diseño. En este capítulo nos centraremos en el diseño del esquema de la base de datos, aunque más adelante en este capítulo se esbozarán brevemente algunas de las otras tareas.

El diseño de un entorno completo de aplicaciones de bases de datos que responda a las necesidades de la empresa que se está modelando exige prestar atención a un amplio conjunto de consideraciones. Estos aspectos adicionales del uso esperado de la base de datos influyen en gran variedad de opciones de diseño en los niveles físico, lógico y de vistas.

7.1.1. Fases del diseño

Para aplicaciones pequeñas puede resultar factible para un diseñador de bases de datos que comprenda los requisitos de la aplicación decidir directamente sobre las relaciones que hay que crear, sus atributos y las restricciones sobre las relaciones. Sin embargo, un proceso de diseño tan directo resulta difícil para las aplicaciones reales, ya que a menudo son muy complejas. Frecuentemente no existe una sola persona que comprenda todas las necesidades de datos de la aplicación. El diseñador de la base de datos debe interactuar con los usuarios para comprender las necesidades de la aplicación; realizar una representación de alto nivel de esas necesidades, que pueda ser comprendida por los usuarios, y luego traducir esos requisitos a niveles inferiores del diseño. Los modelos de datos de alto nivel sirven a los diseñadores de bases de datos ofreciéndoles un marco conceptual en el que especificar de forma sistemática los requisitos de datos de los usuarios de la base de datos y una estructura para la base de datos que satisfaga esos requisitos.

- La fase inicial del diseño de las bases de datos es la caracterización completa de las necesidades de datos de los posibles usuarios de la base de datos. El diseñador de la base de datos debe interactuar intensamente con los expertos y los usuarios del dominio para realizar esta tarea. El resultado de esta fase es una especificación de requisitos del usuario. Aunque existen técnicas para representar en diagramas los requisitos de los usuarios, en este capítulo solo se describirán textualmente esos requisitos.
- A continuación, el diseñador elige el modelo de datos y, aplicando los conceptos del modelo de datos elegido, traduce estos requisitos en un esquema conceptual de la base de datos. El esquema desarrollado en esta fase de **diseño conceptual** proporciona una visión detallada de la empresa. Se suele emplear el modelo entidad-relación, que se estudiará en el resto de este capítulo, para representar el diseño conceptual. En términos del modelo entidad-relación, el esquema conceptual especifica las entidades que se representan en la base de datos, sus atributos, las relaciones entre ellas y las restricciones que las afectan. Generalmente, la fase de diseño conceptual da lugar a la creación de un diagrama entidad-relación que ofrece una representación gráfica del esquema.

  El diseñador revisa el esquema para confirmar que realmente se satisfacen todos los requisitos y que no entran en conflicto entre sí. También puede examinar el diseño para eliminar características redundantes. Su atención en este momento se centra en describir los datos y sus relaciones, más que en especificar los detalles del almacenamiento físico.
- Un esquema conceptual completamente desarrollado indica también los requisitos funcionales de la empresa. En la **especificación de requisitos funcionales** los usuarios describen los tipos de operaciones (o transacciones) que se llevarán a cabo sobre los datos. Algunos ejemplos de operaciones son la modificación o actualización de datos, la búsqueda y recuperación de datos concretos y el borrado de datos. En esta fase de diseño conceptual, el diseñador puede revisar el esquema para asegurarse de que satisface los requisitos funcionales.
- El paso desde el modelo abstracto de datos a la implementación de la base de datos se divide en dos fases de diseño finales.
  - En la **fase de diseño lógico**, el diseñador traduce el esquema conceptual de alto nivel al modelo de datos de la implementación del sistema de bases de datos que se va a usar. El modelo de implementación de los datos suele ser el modelo relacional, y este paso suele consistir en la traducción del esquema conceptual definido mediante el modelo entidad-relación en un esquema de relación.

– Finalmente, el diseñador usa el esquema de base de datos resultante propio del sistema en la siguiente **fase de diseño físico**, en la que se especifican las características físicas de la base de datos. Entre estas características están la forma de organización de los archivos y las estructuras de almacenamiento interno; se estudian en los Capítulos 10 y 11.

El esquema físico de la base de datos puede modificarse con relativa facilidad una vez creada la aplicación. Sin embargo, las modificaciones del esquema lógico suelen resultar más difíciles de llevar a cabo, ya que pueden afectar a varias consultas y actualizaciones dispersas por todo el código de la aplicación. Por tanto, es importante llevar a cabo con cuidado la fase de diseño de la base de datos antes de crear el resto de la aplicación de bases de datos.

7.1.2. Alternativas de diseño

Una parte importante del proceso de diseño de las bases de datos consiste en decidir la manera de representar los diferentes tipos de «cosas», como personas, lugares, productos y similares. Se usa el término *entidad* para hacer referencia a cualquiera de esos elementos claramente identificables. En el ejemplo de base de datos de una universidad, son entidades los profesores, estudiantes, departamentos, asignaturas y oferta de asignaturas.[1] Las diferentes entidades se relacionan entre sí de diversas formas, relaciones que han de ser capturadas en el diseño de la base de datos. Por ejemplo, un estudiante se matricula en una oferta de asignaturas, mientras que un profesor enseña una oferta de asignaturas; matricula y enseña son ejemplos de relaciones entre entidades.

Al diseñar el esquema de una base de datos hay que asegurarse de que se evitan dos peligros importantes:

1. **Redundancia:** un mal diseño puede repetir información. Por ejemplo, si se guarda el identificador de asignatura y el nombre de la asignatura con cada oferta de esta, el nombre se guardaría de forma redundante (es decir, varias veces, de forma innecesaria) con cada oferta de asignaturas. Sería suficiente guardar solo el identificador de asignatura con cada oferta de asignaturas y asociar el nombre con el identificador de la misma una única vez en una entidad asignatura.

   También puede haber redundancia en un esquema relacional. En el ejemplo de la universidad, se tiene una relación de información de sección y una relación distinta con la información de asignatura. Suponga que en su lugar tuviésemos una sola relación en la que se repitiese toda la información de la asignatura (id de asignatura, nombre de la asignatura, nombre del departamento, créditos) para cada sección (oferta) de cada asignatura. Claramente, la información de las asignaturas se guardaría de forma redundante.

   El mayor problema con esta representación redundante de la información es que las copias de una determinada información se pueden convertir en inconsistentes si dicha información se actualiza sin tener la precaución de actualizar todas las copias de la misma. Por ejemplo, las distintas ofertas de asignaturas pueden tener el mismo identificador de asignatura, pero diferentes nombres. No quedaría claro cuál es el nombre correcto de la asignatura. De forma ideal, la información debería estar exclusivamente en un solo lugar.

2. **Incompletitud:** un mal diseño puede hacer que determinados aspectos de la empresa resulten difíciles o imposibles de modelar. Por ejemplo, supóngase que para el caso anterior solo se dispone de entidades para la oferta de asignaturas, sin tener entidades que se correspondan con las asignaturas. De forma equivalente en términos de las relaciones, suponga que tenemos una sola relación en la que se repite toda la información de asignatura para cada sección en que se oferta una asignatura. Sería imposible representar la información sobre una nueva asignatura, a no ser que dicha asignatura esté en una oferta. Se podría intentar solventar la situación guardando valores nulos en la información de sección. Este parche no solo resulta poco atractivo, sino que puede evitarse mediante restricciones de clave primaria.

Evitar los malos diseños no es suficiente. Puede haber gran número de buenos diseños entre los que haya que escoger. Como ejemplo sencillo, considérese un cliente que compra un producto. ¿La venta de este producto es una relación entre el cliente y el producto? Dicho de otra manera, ¿es la propia venta una entidad que está relacionada con el cliente y con el producto? Esta elección, aunque simple, puede suponer una importante diferencia en cuanto a que los aspectos de la empresa se pueden modelar bien. Considerando la necesidad de tomar decisiones como esta para el gran número de entidades y de relaciones que hay en las empresas reales, no es difícil ver que el diseño de bases de datos puede constituir un problema arduo. En realidad, se verá que exige una combinación de conocimientos y de «buen gusto».

7.2. El modelo entidad-relación

El modelo de datos **entidad–relación (E-R)** se desarrolló para facilitar el diseño de bases de datos permitiendo la especificación de un *esquema de la empresa* que representa la estructura lógica global de la base de datos.

El modelo E-R resulta de gran utilidad a la hora de trasladar los significados e interacciones de las empresas del mundo real a esquemas conceptuales. Gracias a esta utilidad, muchas herramientas de diseño de bases de datos permiten dibujar estos conceptos del modelo E-R. El modelo de datos E-R emplea tres conceptos básicos: los conjuntos de entidades, los conjuntos de relaciones y los atributos, que se tratarán en primer lugar. El modelo entidad-relación también tiene asociada una representación en forma de diagramas, los diagramas E-R, que se tratarán más adelante en este capítulo.

7.2.1. Conjuntos de entidades

Una **entidad** es una «cosa» u «objeto» del mundo real que es distinguible de todos los demás objetos. Por ejemplo, cada persona de una universidad es una entidad. Una entidad tiene un conjunto de propiedades, y los valores de algún conjunto de propiedades pueden identificar cada entidad de forma unívoca. Por ejemplo, una persona puede tener una propiedad *persona_id* que identifica unívocamente a dicha persona. Es decir, un valor como 677-89-9011 en *persona_id* identificará de forma única a una persona concreta de la universidad. De forma similar, se puede pensar en las asignaturas como entidades, y *asignatura_id* identifica de forma única a una entidad asignatura en la universidad. Las entidades pueden ser concretas, como las personas o los libros, o abstractas, como las asignaturas, las ofertas de asignaturas o una reserva de un vuelo.

Un **conjunto de entidades** agrupa entidades del mismo tipo que comparten propiedades o atributos. El conjunto de todas las personas que son profesores de una universidad dada, por ejemplo, se puede definir como el conjunto de entidades *profesor*. De forma similar, el conjunto de entidades *estudiante* podría representar al conjunto de todos los estudiantes de la universidad.

En el proceso de modelado normalmente hablamos de *conjunto de entidades* en abstracto, sin aludir a ningún conjunto de entidades individuales. El término **extensión** de un conjunto de entidades se refiere a la colección real de entidades que pertenecen al conjunto

1 Una asignatura puede haberse impartido en varios semestres, así como varias veces en un mismo semestre. Nos referiremos a esta oferta de asignaturas como una sección.

de entidades en cuestión. Es decir, el conjunto de los profesores reales de la universidad forma la extensión del conjunto de entidades *profesor*. La distinción anterior es similar a la diferencia entre una relación y un ejemplar de relación, como se vio en el Capítulo 2.

Los conjuntos de entidades no son necesariamente disjuntos. Por ejemplo, es posible definir el conjunto de entidades de todas las personas de la universidad (*persona*). Una entidad *persona* puede ser una entidad *profesor;* una entidad *estudiante*, ambas o ninguna.

Cada entidad se representa mediante un conjunto de **atributos.** Los atributos son propiedades descriptivas que posee cada miembro de un conjunto de entidades. La designación de un atributo para un conjunto de entidades expresa que la base de datos almacena información parecida relativa a cada entidad del conjunto de entidades; sin embargo, cada entidad puede tener su propio valor para cada atributo. Posibles atributos del conjunto de entidades *profesor* son: *ID*, *nombre*, *nombre_dept* y *sueldo*. En la vida real habría más atributos, como el número de la calle, el número del piso, la provincia, el código postal y el país, pero se omiten para no complicar el ejemplo. Posibles atributos del conjunto de entidades *asignatura* son *asignatura_id*, *nombre_asig*, *nombre_dept* y *créditos*.

Cada entidad tiene un **valor** para cada uno de sus atributos. Por ejemplo, una entidad *profesor* concreta puede tener el valor 12121 para *ID*, el valor Wu para *nombre*, el valor Finanzas para *nombre_dept* y el valor 90000 para *sueldo*.

El atributo *ID* se usa para identificar unívocamente a los profesores, dado que puede haber más de un profesor con el mismo nombre. En Estados Unidos, muchas empresas consideran adecuado usar el número *seguridad_social* de las personas[2] como atributo cuyo valor identifica unívocamente a esa persona. En general, la empresa tendría que crear y asignar un identificador unívoco a cada profesor.

Por tanto, las bases de datos incluyen una serie de conjuntos de entidades, cada una de las cuales contiene cierto número de entidades del mismo tipo. La Figura 7.1 muestra parte de una base de datos de la universidad solo con algunos de los atributos de los dos conjuntos de entidades que se muestran.

Las bases de datos de una universidad pueden incluir otros conjuntos de entidades. Por ejemplo, además del seguimiento de los profesores y de los estudiantes, la universidad también tiene información de las asignaturas, que se representan mediante el conjunto de entidades *asignatura* con los atributos *asignatura_id*, *nombre_asig*, *nombre_dept* y *créditos*. En un entorno real, una base de datos de una universidad puede contener docenas de conjuntos de entidades.

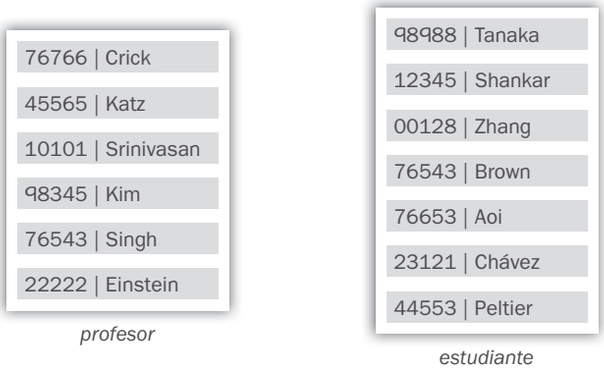


Figura 7.1. Conjuntos de entidades *profesor* y *estudiante*.

2 En los Estados Unidos, el Gobierno asigna a cada persona un número único, denominado número de la seguridad social, para su identificación. Se supone que cada individuo tiene un único número de seguridad social y no hay dos personas que puedan tener el mismo número.

7.2.2. Conjuntos de relaciones

Una **relación** es una asociación entre varias entidades. Por ejemplo, se puede definir una relación *tutor* que asocie al profesor Katz con el estudiante Shankar. Esta relación especifica que Katz es el tutor del estudiante Shankar.

Un **conjunto de relaciones** es un conjunto de relaciones del mismo tipo. Formalmente, es una relación matemática con $n \geq 2$ de conjuntos de entidades (posiblemente no distintos). Si $E_1, E_2 \ldots, E_n$ son conjuntos de entidades, entonces un conjunto de relaciones R es un subconjunto de:

$$\{(e_1, e_2 \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2 \ldots, e_n \in E_n\}$$

donde $(e_1, e_2 \ldots, e_n)$ es una relación.

Considérense las dos entidades *profesor* y *estudiante* de la Figura 7.1. Se define el conjunto de relaciones *tutor* para denotar la asociación entre los profesores y los estudiantes. La Figura 7.2 muestra esta asociación.

Como ejemplo adicional, considérense los dos conjuntos de entidades *estudiante* y *sección*. Se puede definir el conjunto de relaciones *matricula* para denotar la asociación entre un estudiante y las secciones de asignaturas en que ese estudiante está matriculado.

La asociación entre conjuntos de entidades se conoce como *participación*; es decir, los conjuntos de entidades $E_1, E_2 \ldots, E_n$ **participan** en el conjunto de relaciones R. Un **ejemplar de la relación** de un esquema E-R representa una asociación entre las entidades citadas en la empresa real que se está modelando. Como ilustración, la entidad *profesor* Katz, que tiene el identificador *ID* 45565, y la entidad *estudiante* Shankar, que tiene el *ID* 12345, participan en el ejemplar de relación *tutor*. Este ejemplar de relación representa que en la universidad el profesor Katz es el tutor del estudiante Shankar.

La función que desempeña una entidad en una relación se denomina **rol** de esa entidad. Como los conjuntos de entidades que participan en un conjunto de relaciones, generalmente, son distintos, los roles están implícitos y no se suelen especificar. Sin embargo, resultan útiles cuando el significado de una relación necesita aclaración. Tal es el caso cuando los conjuntos de entidades de una relación no son distintos; es decir, el mismo conjunto de entidades participa en un conjunto de relaciones más de una vez, con diferentes roles. En este tipo de conjunto de relaciones, que a veces se denomina conjunto de relaciones **recursivo,** son necesarios los nombres explícitos para los roles con el fin de especificar la manera en que cada entidad participa en cada ejemplar de la relación. Por ejemplo, considérese un conjunto de entidades *asignatura* que almacena información de todas las asignaturas de la universidad. Para indicar la situación donde una asignatura (C2) es un prerrequisito de otra asignatura (C1) se tiene el conjunto de relaciones *prerreq,* que se modela con pares de entidades *asignatura*. La primera asignatura del par tiene el rol de la asignatura C1, mientras que la segunda toma el rol del curso prerrequisito C2. De esta forma, todas las relaciones de *prerreq* se caracterizan por pares (C1, C2); se excluyen los pares (C2, C1).

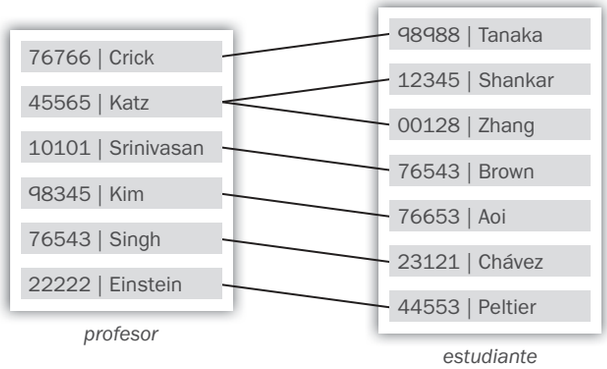


Figura 7.2. Conjunto de relaciones *tutor*.

Una relación puede también tener atributos denominados **atributos descriptivos.** Considérese el conjunto de relaciones *tutor* con los conjuntos de entidades *profesor* y *estudiante*. Se puede asociar el atributo *fecha* con esta relación para especificar la fecha cuando un profesor se convirtió en tutor del estudiante. La relación *tutor* entre las entidades correspondientes al profesor Katz y el estudiante Shankar tiene el valor «10 Junio 2007» en el atributo *fecha*, lo que significa que Katz se convirtió en el tutor de Shankar el 10 de junio de 2007.

La Figura 7.3 muestra el conjunto de relaciones *tutor* con el atributo descriptivo *fecha*. Fíjese que Katz tutela a dos estudiantes con distintas fechas de comienzo.

Como ejemplo adicional de los atributos descriptivos de las relaciones, supóngase que se tienen los conjuntos de entidades *estudiante* y *sección* que participan en el conjunto de relaciones *matriculado*. Puede que se desee almacenar el atributo descriptivo *nota* con la relación, para registrar la nota del estudiante en clase. También se puede guardar un atributo descriptivo *para_nota* si el estudiante se ha matriculado en la asignatura para obtener créditos o solo como oyente.

Cada ejemplar de una relación de un conjunto de relaciones determinado debe identificarse unívocamente a partir de sus entidades participantes, sin usar los atributos descriptivos. Para comprender esto, supóngase que deseemos representar todas las fechas en las que un profesor se ha convertido en tutor de un estudiante concreto. El atributo monovalorado *fecha* solo puede almacenar una fecha. No se pueden representar varias fechas mediante varios ejemplares de la relación entre el mismo profesor y el mismo estudiante, ya que los ejemplares de la relación no quedarían identificados unívocamente usando solo las entidades participantes. La forma correcta de tratar este caso es crear el atributo multivalorado *fecha*, que puede almacenar todas las fechas.

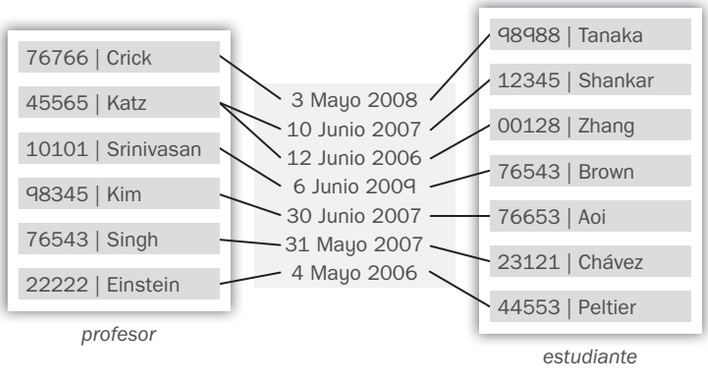


Figura 7.3. *fecha* como atributo del conjunto de relaciones *tutor*.

Puede haber más de un conjunto de relaciones que implique a los mismos conjuntos de entidades. En nuestro ejemplo, los conjuntos de entidades *profesor* y *estudiante* participan en el conjunto de relaciones *tutor*. Además, supóngase que cada estudiante deba tener otro profesor que sirva como tutor del departamento. Entonces los conjuntos de entidades *profesor* y *estudiante* pueden participar en otro conjunto de relaciones: *tutor_dept*.

Los conjuntos de relaciones *tutor* y *tutor_dept* proporcionan un ejemplo de conjunto de relaciones **binario,** es decir, uno que implica dos conjuntos de entidades. La mayor parte de los conjuntos de relaciones de los sistemas de bases de datos son binarios. A veces, no obstante, los conjuntos de relaciones implican a más de dos conjuntos de entidades.

Por ejemplo, considérense los conjuntos de entidades *proyecto*, que representa todos los proyectos de investigación llevados a cabo en la universidad. Considere los conjuntos de entidades *profesor*, *estudiante* y *proyecto*. Cada proyecto puede tener asociados varios profesores y varios estudiantes. Más aún, cada estudiante que trabaja en un proyecto debe tener asociado un profesor que le guía en el proyecto. Por ahora se van a ignorar las primeras dos relaciones, entre proyecto y profesor y entre proyecto y estudiante. Vamos a centrarnos en la información sobre qué profesor está guiando a qué estudiante en un proyecto dado. Para representar esta información, relacionamos los tres conjuntos de entidades mediante el conjunto de relaciones *proy_direc*, que indica qué estudiante concreto participa en un proyecto bajo la dirección de un profesor.

Fíjese que un estudiante podría tener distintos profesores como guías en distintos proyectos, lo que no se puede capturar mediante una relación binaria entre estudiantes y profesores.

El número de conjuntos de entidades que participan en un conjunto de relaciones es también el **grado** de ese conjunto de relaciones. Los conjuntos de relaciones binarios tienen grado 2; los conjuntos de relaciones ternarios tienen grado 3.

7.2.3. Atributos

Para cada atributo hay un conjunto de valores permitidos, denominados **dominio** o **conjunto de valores** de ese atributo. El dominio del atributo *asignatura_id* puede ser el conjunto de todas las cadenas de texto de una cierta longitud. Análogamente, el dominio del atributo *semestre* puede ser el conjunto de todas las cadenas de caracteres del conjunto {Otoño, Invierno, Primavera, Verano}.

Formalmente, cada atributo de un conjunto de entidades es una función que asigna el conjunto de entidades a un dominio. Dado que el conjunto de entidades puede tener varios atributos, cada entidad se puede describir mediante un conjunto de pares (atributo, valor), un par por cada atributo del conjunto de entidades. Por ejemplo, una entidad *profesor* concreta se puede describir mediante el conjunto {(*ID*, 76766), (*nombre*, Crick), (*nombre_dept*, Biología), (*sueldo*, 72000)}, lo que significa que esa entidad describe a una persona llamada Crick cuyo *ID* de profesor es 76766, que es miembro del departamento de Biología con un sueldo de 72.000 €. Ahora se puede ver que existe una integración del esquema abstracto con la empresa real que se está modelando. Los valores de los atributos que describen cada entidad constituyen una parte significativa de los datos almacenados en la base de datos.

Cada atributo, tal y como se usa en el modelo E-R, se puede caracterizar por los siguientes tipos de atributo:

- Atributos **simples** y **compuestos.** En los ejemplos considerados hasta ahora los atributos han sido simples; es decir, no estaban divididos en subpartes. Los atributos **compuestos,** en cambio, se pueden dividir en subpartes (es decir, en otros atributos). Por ejemplo, el atributo *nombre* puede estar estructurado como un atributo compuesto consistente en *nombre*, *primer_apellido* y *segundo_apellido*. Usar atributos compuestos en un esquema de diseño es una buena elección si el usuario desea referirse a un atributo completo en algunas ocasiones y, en otras, solamente a algún componente del atributo. Supóngase que se desease añadir un componente dirección al conjunto de entidades estudiante. La dirección se puede definir como el atributo compuesto *dirección* con los atributos *calle*, *ciudad*, *provincia* y *código postal*.[3] Los atributos compuestos pueden ayudar a agrupar todos los atributos, consiguiendo que los modelos sean más claros.

  Obsérvese también que los atributos compuestos pueden aparecer como una jerarquía. En el atributo compuesto *dirección*, el componente *calle* puede dividirse a su vez en *calle_nombre*, *calle_número* y *piso*. La Figura 7.4 muestra estos ejemplos de atributos compuestos para el conjunto de entidades *profesor*.

3 Supondremos el formato de dirección usado en España y otros muchos países, que incluye un número de código postal.

- Atributos **monovalorados** y **multivalorados.** Todos los atributos que se han especificado en los ejemplos anteriores tienen un único valor para cada entidad concreta. Por ejemplo, el atributo *ID* de *estudiante* para una entidad estudiante específica hace referencia a un único número de *ID* de estudiante. Se dice que estos atributos son **monovalorados.** Puede haber ocasiones en las que un atributo tenga un conjunto de valores para una entidad concreta. Considérese un conjunto de entidades *profesor* con el atributo *número_teléfono*. Cada profesor puede tener cero, uno o varios números de teléfono, y profesores diferentes pueden tener diferente cantidad de teléfonos. Se dice que este tipo de atributo es **multivalorado.** Como ejemplo adicional al conjunto de entidades *profesor* podríamos añadir un atributo que agrupara la lista de *nombre_dependiente*, con todos los que dependen de él. Este atributo es multivalorado, ya que para un profesor dado puede tener cero, uno o más dependiendo de él.

  Para indicar que un atributo es multivalorado se encierra entre llaves, por ejemplo {*número_teléfono*} o {*nombre_dependiente*}. Si resulta necesario, se pueden establecer apropiadamente límites inferior y superior al número de valores en el atributo **multivalorado.** Por ejemplo, una universidad puede limitar a dos el número de teléfonos que se guardan por profesor. El establecimiento de límites en este caso expresa que el atributo *número_teléfono* del conjunto de entidades *profesor* puede tener entre cero y dos valores.
- Atributos **derivados.** El valor de este tipo de atributo se puede obtener a partir del valor de otros atributos o entidades relacionados. Por ejemplo, suponga que el conjunto de entidades *profesor*, que tiene un atributo *estudiantes_tutelados,* representa el número de estudiantes que tiene como tutor. Ese atributo se puede obtener contando el número de entidades *estudiante* asociadas con ese profesor.

  Como ejemplo adicional, supóngase que el conjunto de entidades *profesor* tiene el atributo *edad*, que indica la edad del profesor. Si el conjunto de entidades *profesor* tiene también un atributo *fecha_de_nacimiento*, se puede calcular *edad* a partir de *fecha_de_nacimiento* y de la fecha actual. Por tanto, *edad* es un atributo derivado. En este caso, *fecha_de_nacimiento* puede considerarse un atributo *básico*, o *almacenado*. El valor de los atributos derivados no se almacena, sino que se calcula cada vez que hace falta.

Los atributos toman valores **nulos** cuando las entidades no tienen ningún valor para ese atributo. El valor *nulo* también puede indicar «no aplicable» , es decir, que el valor no existe para esa entidad. Por ejemplo, una persona puede no tener un segundo apellido (si no es español). *Nulo* puede también designar que el valor del atributo es desconocido. Un valor desconocido puede ser *falta* (el valor existe pero no se tiene esa información) o *desconocido* (no se sabe si ese valor existe realmente o no).

Por ejemplo, si el valor *nombre* de un *profesor* dado es *nulo*, se da por supuesto que el valor falta, ya que todos los profesores deben tener nombre. Un valor *nulo* para el atributo *piso* puede significar que la dirección no incluye un piso (no aplicable), que existe el valor piso pero no se conoce cuál es (falta), o que no se sabe si el valor piso forma parte o no de la dirección del profesor (desconocido).

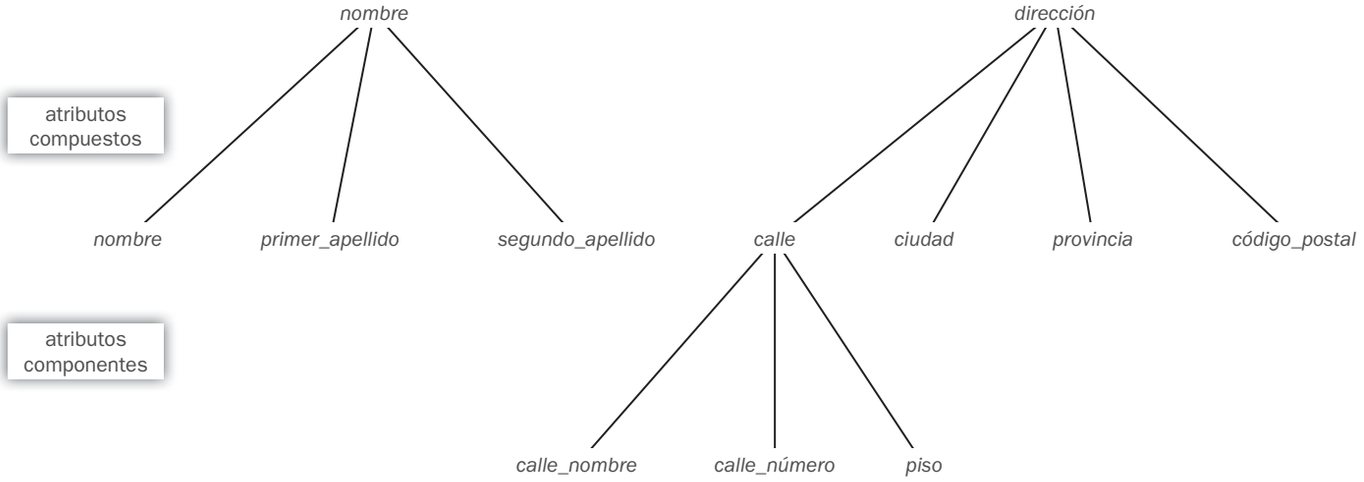


Figura 7.4. Atributos compuestos de profesor *nombre* y *dirección*.

7.3. Restricciones

Un esquema de empresa E-R puede definir ciertas restricciones que el contenido de la base de datos debe respetar. En este apartado se examinan la correspondencia de cardinalidades y las restricciones de participación

7.3.1. Correspondencia de cardinalidades

La **correspondencia de cardinalidades,** o razón de cardinalidad, expresa el número de entidades a las que otra entidad se puede asociar mediante un conjunto de relaciones.

La correspondencia de cardinalidades resulta muy útil para describir conjuntos de relaciones binarias, aunque pueda contribuir a la descripción de conjuntos de relaciones que impliquen más de dos conjuntos de entidades. En este apartado se centrará la atención únicamente en los conjuntos de relaciones binarias.

Para un conjunto de relaciones binarias R entre los conjuntos de entidades A y B, la correspondencia de cardinalidades debe ser una de las siguientes:

- **Uno a uno.** Cada entidad de A se asocia, *a lo sumo*, con una entidad de B, y cada entidad de B se asocia, *a lo sumo*, con una entidad de A (véase la Figura 7.5a).
- **Uno a varios.** Cada entidad de A se asocia con cualquier número (cero o más) de entidades de B. Cada entidad de B, sin embargo, se puede asociar, *a lo sumo*, con una entidad de A (véase la Figura 7.5b).

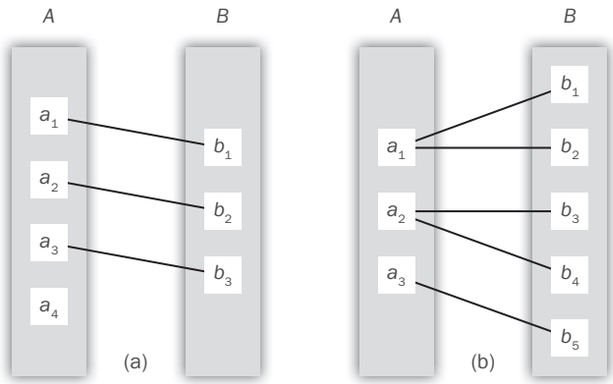


Figura 7.5. Correspondencia de cardinalidades (a) Uno a uno. (b) Uno a varios.

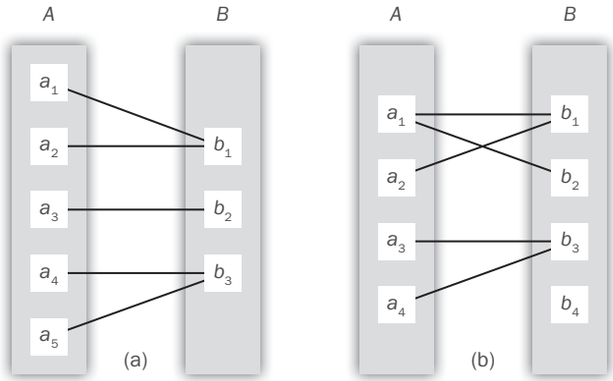


Figura 7.6. Correspondencia de cardinalidades. (a) Varios a uno. (b) Varios a varios.

- **Varios a uno.** Cada entidad de A se asocia, *a lo sumo*, con una entidad de B. Cada entidad de B, sin embargo, se puede asociar con cualquier número (cero o más) de entidades de A (véase la Figura 7.6a).
- **Varios a varios.** Cada entidad de A se asocia con cualquier número (cero o más) de entidades de B, y cada entidad de B se asocia con cualquier número (cero o más) de entidades de A (véase la Figura 7.6b).

La correspondencia de cardinalidades adecuada para un conjunto de relaciones dado depende, obviamente, de la situación del mundo real que el conjunto de relaciones modele.

Como ilustración, considere el conjunto de relaciones *tutor*. Si, en una universidad dada, cada estudiante solo puede tener como tutor a un profesor y cada profesor puede tutelar a varios estudiantes, entonces el conjunto de relaciones de *profesor* a *estudiante* es uno a varios. Si un estudiante puede ser tutelado por varios profesores (como en el caso de tutela compartida), el conjunto de relaciones es varios a varios.

7.3.2. Restricciones de participación

Se dice que la participación de un conjunto de entidades E en un conjunto de relaciones R es **total** si cada entidad de E participa, al menos, en una relación de R. Si solo algunas entidades de E participan en relaciones de R, se dice que la participación del conjunto de entidades E en la relación R es **parcial.** En la Figura 7.5a, la participación de B en el conjunto de relaciones es total, mientras que la participación de A en el conjunto de relaciones es parcial. En la Figura 7.5b la participación de ambos, tanto A como B, en el conjunto de relaciones es total.

Por ejemplo, se puede esperar que cada entidad *estudiante* esté relacionada al menos con un profesor mediante la relación *tutor*. Por tanto, la participación de *estudiante* en el conjunto de relaciones *tutor* es total. En cambio, un profesor puede no ser tutor de ningún estudiante. Por tanto, es posible que solo algunas de las entidades *profesor* estén relacionadas con el conjunto de entidades *estudiante* mediante la relación *tutor*, y la participación de *profesor* en la relación *tutor* es, por tanto, parcial.

7.3.3. Claves

Es necesario tener una forma de especificar la manera de distinguir las entidades pertenecientes a un conjunto de entidades dado. Conceptualmente, cada entidad es distinta; desde el punto de vista de las bases de datos, sin embargo, la diferencia entre ellas se debe expresar en términos de sus atributos.

Por lo tanto, los valores de los atributos de cada entidad deben ser tales que permitan *identificar unívocamente* a esa entidad. En otras palabras, no se permite que ningún par de entidades de un conjunto de entidades tenga exactamente el mismo valor en todos sus atributos.

La noción de una *clave* para un esquema de relación, tal como se definió en la Sección 2.3, aplica directamente a los conjuntos de entidades. Es decir, una clave para una entidad es un conjunto de atributos que es suficiente para distinguir unas entidades de otras. Los conceptos de superclave, clave candidata y clave primaria se aplican a los conjuntos de entidades solo si son aplicables a los esquemas de relación.

Las claves también ayudan a identificar relaciones de forma única y, por tanto, permiten distinguir unas relaciones de otras. A continuación se definen las nociones correspondientes de claves para las relaciones.

La clave primaria de un conjunto de entidades nos permite distinguir entre varias entidades del conjunto. Necesitamos un mecanismo similar para distinguir entre las distintas relaciones de un conjunto de relaciones.

Sea R un conjunto de relaciones que implica a los conjuntos de entidades E_1 E_2 …, E_n. Por ejemplo, *clave-primaria* (E_i) indica el conjunto de atributos que forman la clave primaria de la entidad E_i . Suponga que los nombres de los atributos de todas las claves primarias son únicos. La composición de las claves primarias para un conjunto de relaciones depende del conjunto de atributos asociados con el conjunto de relaciones R.

Si el conjunto de relaciones R no tiene atributos asociados, entonces el conjunto de atributos

$$\textit{clave-primaria}(E_1) \cup \textit{clave-primaria}(E_2) \cup \ldots \cup \textit{clave-primaria}(E_n)$$

describe una relación individual en el conjunto R.

Si el conjunto de relaciones R tiene los atributos a_1 , a_2 …, a_m asociados, entonces el conjunto de atributos

$$\textit{clave-primaria}\ (E_1) \cup \textit{clave-primaria}\ (E_2) \cup \ldots \cup \textit{clave-primaria}(E_n) \cup \{a_1, a_2 \ldots, a_m\}$$

describe una relación individual en el conjunto R.

En ambos casos, el conjunto de atributos

$$\textit{clave-primaria}\ (E_1) \cup \textit{clave-primaria}\ (E_2) \cup \ldots \cup \textit{clave-primaria}\ (E_n)$$

forma una superclave del conjunto de relaciones.

Si los nombres de los atributos de las claves primarias no son únicos entre los conjuntos de entidades, los atributos se renombran para distinguirlos; el nombre del conjunto de entidades junto con el nombre del atributo formarán un nombre único. Si un conjunto de entidades participa más de una vez en un conjunto de relaciones (como en la relación *prerreq*, en la Sección 7.2.2), en su lugar se usa el rol del nombre, en lugar del nombre del conjunto de entidades, para formar un nombre de atributo que sea único.

La estructura de las claves primarias de un conjunto de relaciones depende de la correspondencia de cardinalidad del conjunto de relaciones. Como ejemplo, suponga el conjunto de entidades *profesor* y *estudiante*, y el conjunto de relaciones *tutor*, con el atributo *fecha*, de la Sección 7.2.2. Suponga que el conjunto de relaciones es de varios a varios. Entonces la clave primaria de *tutor* consistirá en la unión de las claves primarias de *profesor* y *estudiante*. Si la relación es de varios a uno de *estudiante* a *profesor;* es decir, cada estudiante puede tener como mucho un tutor, entonces la clave primaria de *tutor* es simplemente la clave primaria de *estudiante*. Sin embargo, si un profesor puede tutelar solo a un estudiante; es decir, si la relación *tutor* es varios a uno de *profesor* a *estudiante*, entonces la clave primaria de *tutor* es simplemente la clave primaria de *profesor*. Para las relaciones uno a uno se puede usar como clave primaria cualquiera de las claves candidatas.

Para las relaciones no binarias, si no existen restricciones de cardinalidad entonces la superclave formada como se ha descrito anteriormente en esta sección es la única clave candidata, y se elige como clave primaria. La elección de la clave primaria es más complicada si existen restricciones de cardinalidad. Como no se ha tratado cómo especificar las restricciones de cardinalidad para relaciones no binarias, no se va a tratar este tema en este capítulo. Se considerará con más detalle en las Secciones 7.5.5 y 8.4.

7.4. Eliminar atributos redundantes de un conjunto de entidades

Cuando se diseña una base de datos usando el modelo E-R, normalmente se comienza identificando los conjuntos de entidades que deberían incluirse. Por ejemplo, en la organización de la universidad que ha tratado hasta ahora, se decidió incluir entidades como *estudiante*, *profesor*, etc. Una vez decididos los conjuntos de entidades hay que elegir sus atributos. Estos atributos se supone que representan los distintos valores que se desea recoger en la base de datos. Para la organización de la universidad se decidió que el conjunto de entidades *profesor* incluyese los atributos *ID*, *nombre*, *nombre_dept* y *sueldo*. Podrían haberse incluido atributos como *número_teléfono*, *número_despacho*, *web_personal*, etc. La elección sobre qué atributos incluir es decisión del diseñador, que tiene el conocimiento adecuado de la estructura de la empresa.

Una vez elegidas las entidades y sus correspondientes atributos, se forman los conjuntos de relaciones entre las distintas entidades. Estos conjuntos de relaciones pueden llevar a situaciones en las que los atributos de algunos conjuntos de entidades sean redundantes y se necesite eliminarlos de los conjuntos de entidades originales. Para mostrarlo, suponga los conjuntos de entidades *profesor* y *departamento*:

- El conjunto de entidades *profesor* incluye los atributos *ID*, *nombre*, *nombre_dept*, y *sueldo*, siendo *ID* la clave primaria.
- El conjunto de entidades *departamento* incluye los atributos *nombre_dept*, *edificio* y *presupuesto*, siendo *nombre_dept* la clave primaria.

Se modela que cada profesor tiene asociado un departamento utilizando el conjunto de relaciones *profesor_dept* relacionando *profesor* y *departamento*.

El atributo *nombre_dept* aparece en ambos conjuntos de entidades. Como es la clave primaria del conjunto de entidades *departamento*, es redundante en el conjunto de entidades *profesor* y habría que eliminarla.

Eliminar el atributo *nombre_dept* del conjunto de entidades *profesor* parece ser un tanto contraintuitivo, ya que la relación *profesor* que se utilizó en el capítulo anterior tenía un atributo *nombre_dept*. Como se verá más adelante, cuando se crea un esquema relacional de un diagrama E-R, el atributo *nombre_dept* de hecho se añade a la relación *profesor*, pero solo si cada profesor tiene asociado como mucho un departamento. Si un profesor tiene asociado más de un departamento, la relación entre profesores y departamentos se registra en una relación distinta *profesor_dept*.

Al realizar la conexión entre profesores y departamentos de manera uniforme como una relación, en lugar de como un atributo de *profesor*, consigue que la relación sea explícita y ayuda a evitar una suposición prematura de que cada profesor está asociado solo con un departamento.

De manera similar, el conjunto de entidades *estudiante* está relacionado con el conjunto de entidades *departamento* mediante la relación *estudiante_dept*, y por tanto no es necesario el atributo *nombre_dept* en *estudiante*.

Como ejemplo adicional, suponga la oferta de cursos (secciones) junto con las franjas horarias de dicha oferta. Cada franja horaria viene identificada con un *franja_horaria_id*, y tiene asociadas un conjunto de horas semanales, cada una identificada por el día de la semana, la hora de inicio y la hora de finalización. Se decide modelar el conjunto de horas semanales como un atributo multivalorado. Suponga que se modelan los conjuntos de entidades *sección* y *franja horaria* de la siguiente forma:

- El conjunto de entidades *sección* incluye los atributos *asignatura_id*, *secc_id*, *semestre*, *año*, *edificio*, *aula* y *franja_horaria_id*, con (*asignatura_id*, *secc_id*, *año*, *semestre*) como clave primaria.
- El conjunto de entidades *franja_horaria* incluye los atributos *franja_horaria_id*, que es la clave primaria,[4] y un atributo compuesto multivalorado {(*día*, *hora_inicio*, *hora_fin*)}.[5]

Estas entidades se relacionan mediante el conjunto de relaciones *secc_franja_horaria*.

El atributo *franja_horaria_id* aparece en ambos conjuntos de entidades. Como es clave primaria en el conjunto de entidades *franja_horaria*, resulta redundante en el conjunto de entidades *sección* y, por tanto, hay que eliminarlo.

Como ejemplo final, suponga que se tiene un conjunto de entidades *aula*, con los atributos *edificio*, *núm_aula* y *capacidad*, siendo *edificio* y *núm_aula* las claves primarias. Suponga también que se desea establecer un conjunto de relaciones *secc_aula* que relaciona *sección* con *aula*. Entonces los atributos {*edificio*, *núm_aula*} resultan redundantes en el conjunto de entidades *sección*.

En un buen diseño entidad-relación no existen atributos redundantes. Para el ejemplo de la universidad, a continuación se da la lista de conjuntos de entidades y sus atributos, donde se subrayan las claves primarias.

- **aula:** con atributos (*edificio*, *núm_aula*, *capacidad*).
- **departamento:** con atributos (*nombre_dept*, *edificio*, *presupuesto*).
- **asignatura:** con atributos (*asignatura_id*, *nombre_asig*, *créditos*).
- **profesor:** con atributos (*ID*, *nombre*, *sueldo*).
- **sección:** con atributos (*asignatura_id*, *secc_id*, *semestre*, *año*).
- **estudiante:** con atributos (*ID*, *nombre*, *tot_créd*).
- **franja_horaria:** con atributos (*franja_horaria_id*, {(*día*, *hora_inicio*, *hora_fin*)}).

4 Más adelante se verá que para la clave primaria de la relación creada del conjunto de entidades *franja_horaria* se incluye *día* y *hora_inicio*; sin embargo, *día* y *hora_inicio* no forman parte de la clave primaria del conjunto de entidades *franja_horaria*.

5 Opcionalmente, podríamos darle un nombre como *reunión* al atributo compuesto que contiene *día*, *hora_inicio* y *hora_fin*.

Los conjuntos de relaciones del diseño son los siguientes:

- **profesor_dept:** relaciona profesores con departamentos.
- **estudiante_dept:** relaciona estudiantes con departamentos.
- **enseña:** relaciona profesores con secciones.
- **matricula:** relaciona estudiantes con secciones, con el atributo descriptivo *nota*.
- **asignatura_dept:** relaciona asignaturas con departamentos.
- **secc_asignatura:** relaciona secciones con asignaturas.
- **secc_aula:** relaciona secciones con aulas.
- **secc_franja_horaria:** relaciona secciones con franjas horarias.
- **tutor:** relaciona estudiantes con profesores.
- **prerreq:** relaciona asignaturas con prerrequisitos de asignaturas.

Se puede verificar que ninguno de los conjuntos de entidades tiene elementos redundantes. Más aún, se puede comprobar que toda la información (distinta de las restricciones) del esquema relacional de la bases de datos de la universidad, que se vio anteriormente en la Figura 2.8 del Capítulo 2, queda recogida en el diseño anterior, pero con varios atributos del diseño relacional que se han sustituido por relaciones en el diseño E-R.

7.5. Diagramas entidad-relación

Como se vio brevemente en la Sección 1.3.3, los **diagramas E-R** pueden expresar gráficamente la estructura lógica general de las bases de datos. Los diagramas E-R son sencillos y claros, cualidades que pueden ser responsables en gran parte de la popularidad del modelo E-R.

7.5.1. Estructura básica

Un diagrama E-R consta de los siguientes componentes principales:

- **Rectángulos divididos en dos partes,** que representan conjuntos de entidades. La primera parte, que en este libro aparece con fondo gris, contiene el nombre de los conjuntos de entidades. La segunda parte contiene los nombres de todos los atributos del conjunto de entidades.
- **Rombos,** que representan conjuntos de relaciones.
- **Rectángulos sin dividir**, que representan los atributos de un conjunto de relaciones. Los atributos que forman parte de la clave primaria aparecen subrayados.
- **Líneas,** que unen conjuntos de entidades con conjuntos de relaciones.
- **Líneas discontinuas,** que unen atributos de un conjunto de relaciones con conjuntos de relaciones.
- **Líneas dobles,** que indican la participación total de una entidad en un conjunto de relaciones.
- **Rombos dobles,** que representan conjuntos de relaciones identificadas que se unen a conjuntos de entidades débiles (los conjuntos de relaciones identificadas y los conjuntos de entidades débiles se verán más adelante en la Sección 7.5.6).

Considere el diagrama E-R de la Figura 7.7, que consta de dos conjuntos de entidades, *profesor* y *estudiante* relacionadas mediante el conjunto de relaciones *tutor*. Los atributos asociados con *profesor* son *ID*, *nombre* y *sueldo*. Los atributos asociados con *estudiante* son *ID*, *nombre* y *tot_créd*. En la Figura 7.7, los atributos del conjunto de entidades que son miembros de la clave primaria se pueden ver subrayados.

Si un conjunto de relaciones tiene algunos atributos asociados, entonces dichos atributos se encierran en un rectángulo y se une el rectángulo con una línea discontinua al rombo que representa el conjunto de relaciones. Por ejemplo, en la Figura 7.8 se tiene el atributo descriptivo *fecha* asociado con el conjunto de relaciones *tutor* para indicar la fecha en que el profesor se convirtió en tutor.

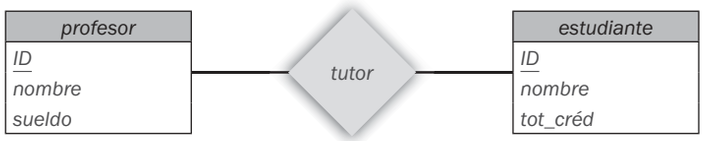


Figura 7.7. Diagrama E-R correspondiente a profesores y estudiantes.

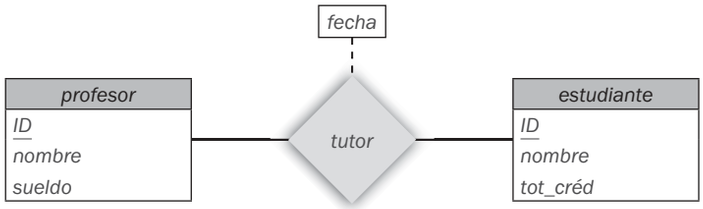


Figura 7.8. Diagrama E-R con un atributo asociado al conjunto de relaciones.

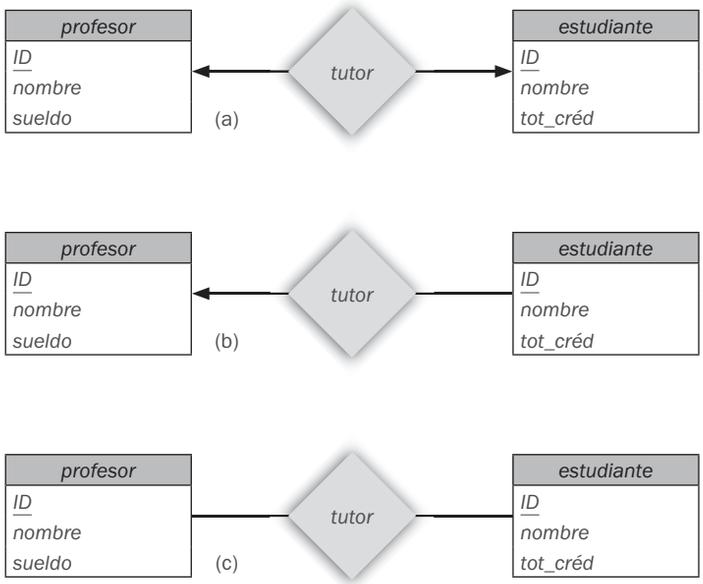


Figura 7.9. Relaciones. (a) Uno a uno. (b) Uno a varios. (c) Varios a varios.

7.5.2. Correspondencia de cardinalidades

El conjunto de relaciones *tutor*, entre los conjuntos de entidades *profesor* y *estudiante*, puede ser uno a uno, uno a varios, varios a uno o varios a varios. Para distinguir entre estos tipos, se dibuja una línea dirigida (→) o una línea sin dirección (—) entre el conjunto de relaciones y el conjunto de entidades dado, de la siguiente forma:

- **Uno a uno:** se dibuja una línea dirigida desde el conjunto de relaciones *tutor* tanto al conjunto de entidades *profesor* como a *estudiante* (véase la Figura 7.9a). Esto indica que un profesor puede ser tutor de, como mucho, un estudiante, y un estudiante puede tener, como mucho, un tutor.
- **Uno a varios:** se dibuja una línea dirigida desde el conjunto de relaciones *tutor* al conjunto de entidades *profesor* y una línea sin dirección al conjunto de entidades *estudiante* (véase la Figura 7.9b). Esto indica que un profesor puede ser tutor de varios estudiantes, pero un estudiante puede tener, como mucho, un tutor.

- **Varios a uno:** se dibuja una línea sin dirección desde el conjunto de relaciones *tutor* al conjunto de entidades *profesor* y una línea dirigida al conjunto de entidades *estudiante*. Esto indica que un profesor puede ser tutor de, como mucho, un estudiante, pero un estudiante puede tener varios tutores.
- **Varios a varios:** se dibuja una línea sin dirección desde el conjunto de relaciones *tutor* a los conjuntos de entidades *profesor* y *estudiante* (véase la Figura 7.9c). Esto indica que un profesor puede ser tutor de varios estudiantes y un estudiante puede tener varios tutores.

Los diagramas E-R también proporcionan una forma de indicar restricciones más complejas sobre el número de veces que una entidad participa en una relación de un conjunto de relaciones. Una línea puede tener asociada una cardinalidad mínima y máxima, de la forma $l \ldots h$, donde l es el mínimo y h el máximo de cardinalidad. Un valor mínimo de 1 indica la participación total del conjunto de entidades en el conjunto de relaciones; es decir, todas las entidades del conjunto de entidades tienen al menos una relación con el conjunto de relaciones. Un valor máximo de 1 indica que la entidad participa, como mucho, en una relación, mientras que un valor máximo de $*$ indica que no existe límite.

Por ejemplo, considere la Figura 7.10. La línea entre *tutor* y *estudiante* tiene una restricción de cardinalidad de 1…1, lo que significa que el mínimo y el máximo de cardinalidad es 1. Es decir, todos los estudiantes tienen que tener exactamente un tutor. El límite 0…$*$ en la línea entre *tutor* y *profesor* indica que un profesor puede tener cero o más estudiantes. Es decir, la relación *tutor* es uno a varios de *profesor* a *estudiante* y, por tanto, la participación de *estudiante* en *tutor* es una relación total, lo que implica que un estudiante tiene que tener un tutor.

Resulta fácil malinterpretar el 0…$*$ de la parte izquierda y pensar que la relación *tutor* es de varios a uno desde *profesor* a *estudiante*, que es exactamente la interpretación contraria a la correcta.

Si en ambos lados el valor máximo es de 1, la relación es uno a uno. Si se hubiese especificado un límite de cardinalidad de 1…$*$ en el lado izquierdo se podría decir que los profesores deben tutelar al menos a un estudiante.

El diagrama E-R de la Figura 7.10 se podría haber dibujado de forma alternativa con una línea doble de *estudiante* a *tutor* y con una línea dirigida de *tutor* a *profesor*, en lugar de indicar las restricciones de cardinalidad. Este diagrama alternativo indica exactamente las mismas restricciones que las indicadas en la figura.

7.5.3. Atributos complejos

En la Figura 7.11 se muestra cómo se pueden representar los atributos compuestos en la notación E-R. En este caso, un atributo compuesto nombre, con los componentes *nombre*, *primer_apellido* y *segundo_apellido* sustituye al atributo simple *nombre* de *profesor*. Como ejemplo adicional, suponga que se desea añadir una dirección al conjunto de entidades *profesor*. La dirección se define como un atributo compuesto *dirección* con los atributos *calle*, *ciudad*, *provincia* y *código postal*. El atributo *calle* a su vez es un atributo compuesto cuyos atributos son *calle_nombre*, *calle_número* y *piso*.

En la Figura 7.11 también se muestra un atributo multivalorado *número_teléfono*, indicado por «{*número_teléfono*}», y un atributo derivado *edad*, indicado con «*edad*( )».

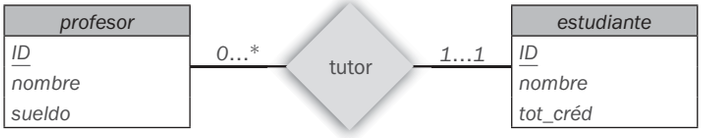


Figura 7.10. Límites de cardinalidad en los conjuntos de relaciones.

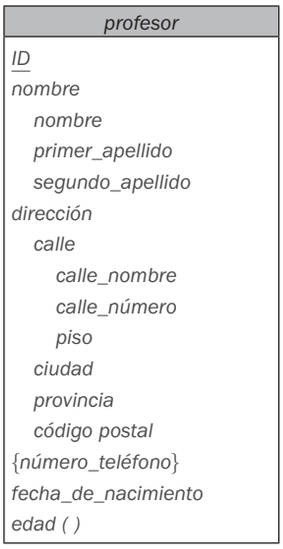


Figura 7.11. Diagrama E-R con atributos compuestos, multivalorados y derivados.

7.5.4. Roles

Los roles se indican en un diagrama E-R etiquetando las líneas que conectan los rombos con los rectángulos. En la Figura 7.12 se muestran los roles *asignatura_id* y *prerreq_id* entre el conjunto de entidades *asignatura* y el conjunto de relaciones *prerreq*.

7.5.5. Conjunto de relaciones no binarias

En un diagrama E-R se pueden especificar fácilmente conjuntos de relaciones no binarias. En la Figura 7.13 se pueden ver tres conjuntos de entidades *profesor*, *estudiante* y *proyecto*, relacionados mediante el conjunto de relaciones *proy_direc*.

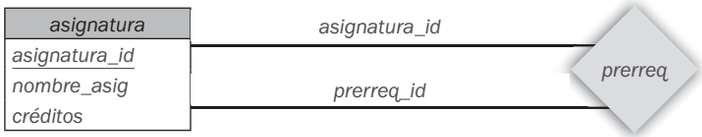


Figura 7.12. Diagrama E-R con indicadores de roles.

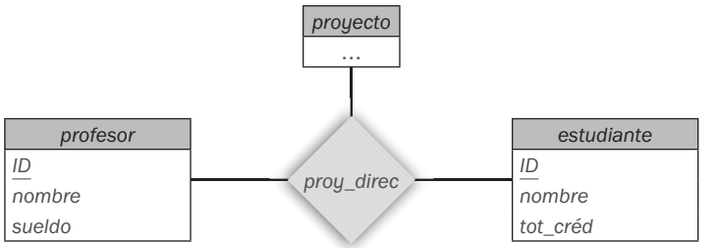


Figura 7.13. Diagrama E-R con una relación ternaria.

En el caso de conjuntos de relaciones no binarias se pueden especificar algunos tipos de relaciones varios a uno. Supóngase que un *estudiante* puede tener, a lo sumo, un profesor como director de un proyecto. Esta restricción se puede especificar mediante una flecha que apunte a *profesor* en la línea desde *proy_direc*.

Como máximo se permite una flecha desde cada conjunto de relaciones, ya que los diagramas E-R con dos o más flechas salientes de cada conjunto de relaciones no binarias se pueden interpretar de dos formas. Suponga que hay un conjunto de relaciones R entre los conjuntos de entidades $A_1, A_2 \ldots, A_n$ y que las únicas flechas están en las líneas hacia los conjuntos de entidades $A_{i+1}, A_{i+2} \ldots, A_n$.

Entonces, las dos interpretaciones posibles son:

1. Una combinación concreta de entidades de $A_1, A_2 \ldots, A_i$ se puede asociar, a lo sumo, con una combinación de entidades de A_{i+1}, $A_{i+2} \ldots, A_n$. Por tanto, la clave primaria de la relación R se puede crear mediante la unión de las claves primarias de $A_1, A_2 \ldots, A_i$.
2. Para cada conjunto de entidades A_k , $i < k \leq n$, cada combinación de las entidades de los otros conjuntos de entidades se puede asociar, a lo sumo, con una entidad de A_k . Cada conjunto $\{A_1, A_2 \ldots, A_{k-1}, A_{k+1} \ldots, A_n\}$, para $i < k \leq n$, forma, entonces, una clave candidata.

Cada una de estas interpretaciones se ha usado en diferentes libros y sistemas. Para evitar confusiones, solo se permite una flecha saliente de cada conjunto de relaciones, en cuyo caso las dos interpretaciones son equivalentes. En el Capítulo 8 (Sección 8.4) se estudia el concepto de *dependencia funcional*, que permite especificar cualquiera de estas dos interpretaciones sin ambigüedad.

7.5.6. Conjunto de entidades débiles

Considere una entidad *sección* identificada unívocamente por un identificador de asignatura, semestre, año e identificador de sección. Claramente, las entidades sección están relacionadas con las entidades asignatura. Suponga que se crea un conjunto de relaciones *secc_asignatura* entre los conjuntos de entidades *sección* y *asignatura*.

Ahora, observe que la información en *secc_asignatura* es redundante, puesto que *sección* ya tiene un atributo *asignatura_id* que identifica el curso con el que dicha asignatura está relacionada. Una opción para combatir esta redundancia es evitar la relación *secc_asignatura*; sin embargo, haciendo esto la relación entre *sección* y *asignatura* se convierte en implícita mediante un atributo, lo que no es deseable.

Una forma alternativa de tratar con esta redundancia es no guardar el atributo *asignatura_id* en la entidad *sección* y guardar solo el resto de los atributos *secc_id*, *año* y *semestre*.[6] Sin embargo, el conjunto de entidades *sección* no tiene entonces suficientes atributos para identificar una entidad *sección* de forma unívoca; aunque cada entidad *sección* es única, las secciones para diferentes asignaturas pueden compartir el mismo *secc_id*, *año* y *semestre*. Para tratar con este problema se crea la relación *secc_asignatura* como una relación especial que proporciona información extra, en este caso el *asignatura_id*, requerido para identificar las entidades *sección* de forma única.

La noción de *conjunto de entidades débil* formaliza la intuición anterior. Se denomina **conjunto de entidades débiles** a los conjuntos de entidades que no tienen suficientes atributos para formar una clave primaria. Los conjuntos de entidades que tienen una clave primaria se denominan **conjuntos de entidades fuertes.**

Para que un conjunto de entidades débiles tenga sentido debe estar asociado con otro conjunto de entidades, denominado **conjunto de entidades identificadoras** o **propietarias.** Cada entidad débil debe asociarse con una entidad identificadora; es decir, se dice que el conjunto de entidades débiles **depende existencialmente** del conjunto de entidades identificadoras. Se dice que el conjunto de entidades identificadoras es **propietario** del conjunto de entidades débiles al que identifica. La relación que asocia el conjunto de entidades débiles con el conjunto de entidades identificadoras se denomina **relación identificadora.**

La relación identificadora es varios a uno del conjunto de entidades débiles al conjunto de entidades identificadoras, y la participación del conjunto de entidades débiles en la relación es total. El conjunto de relaciones identificadoras no debería tener atributos descriptivos, ya que cualquiera de estos atributos se puede asociar con el conjunto de entidades débiles.

En nuestro ejemplo, el conjunto de entidades identificadoras para *sección* es *asignatura*, y la relación *secc_asignatura*, que asocia las entidades *sección* con sus correspondientes entidades *asignatura*, es la relación identificadora.

Aunque los conjuntos de entidades débiles no tienen clave primaria, hace falta un medio para distinguir entre todas las entidades del conjunto de entidades débiles que dependen de una entidad fuerte concreta. El **discriminador** de un conjunto de entidades débiles es un conjunto de atributos que permite que se haga esta distinción. Por ejemplo, el discriminador del conjunto de entidades débiles *sección* consta de los atributos *secc_id*, *año* y *semestre*, ya que, para cada asignatura, este conjunto de atributos identifica de forma única una única sección para dicha asignatura. El discriminador del conjunto de entidades débiles se denomina *clave parcial* del conjunto de entidades.

La clave primaria de un conjunto de entidades débiles se forma con la clave primaria del conjunto de entidades identificadoras y el discriminador del conjunto de entidades débiles. En el caso del conjunto de entidades *sección*, su clave primaria es {*asignatura_id*, *secc_id*, *año*, *semestre*}, donde *asignatura_id* es la clave primaria del conjunto de entidades identificadoras; es decir, *asignatura* y {*secc_id*, *año*, *semestre*} distingue las entidades *sección* de una misma asignatura.

Se podría haber elegido que *secc_id* fuera globalmente único entre todas las asignaturas que se ofertan en la universidad, en cuyo caso el conjunto de entidades *sección* tendría una clave primaria. Sin embargo, conceptualmente una *sección* seguiría dependiendo de una *asignatura* para que exista, lo que se hace explícito constituyéndola como conjunto de entidades débiles.

En los diagramas E-R, un conjunto de entidades débiles se dibuja con un rectángulo, como un conjunto de entidades fuerte, pero con dos diferencias:

- El discriminador de una entidad débil se subraya con una línea discontinua en lugar de con una continua.
- El conjunto de relaciones que conecta los conjuntos de entidades débiles con el conjunto de entidades fuertes identificadoras se dibuja con un rombo doble.

En la Figura 7.14, el conjunto de entidades débiles *sección* depende del conjunto de entidades fuertes *asignatura* mediante el conjunto de relaciones *secc_asignatura*.

La figura también muestra el uso de las líneas dobles para indicar *participación total*; la participación de un conjunto de entidades (débiles) *sección* en la relación *secc_asignatura* es total, lo que significa que todas las secciones deben estar relacionadas a través de *secc_asignatura* con alguna asignatura. Finalmente, la flecha desde *secc_asignatura* a *asignatura* indica que todas las secciones están relacionadas con una única asignatura.

Los conjuntos de entidades débiles pueden participar en otras relaciones, aparte de en la relación identificadora. Por ejemplo, la entidad *sección* puede participar en una relación con el conjunto de entidades *franja_horaria*, identificando los tiempos en que tiene lugar la reunión para una determinada sección de asignatura. Los conjuntos de entidades débiles pueden participar como propietarios de una relación identificadora con otro conjunto de entidades débiles. También es posible tener conjuntos de entidades débiles con más de un conjunto de entidades identificadoras. Cada entidad débil se identificaría mediante una combinación de entidades, una de cada conjunto de entidades identificadoras. La clave primaria de la entidad débil consistiría en la unión de las claves primarias de los conjuntos de entidades identificadoras y el discriminador del conjunto de entidades débiles.

6 Tenga en cuenta que el esquema relacional que eventualmente se cree del conjunto de entidades *sección* no tiene el atributo *asignatura_id*, por las razones que se verán más claras posteriormente, aunque se haya eliminado el atributo *asignatura_id* del conjunto de entidades *sección*.

Figura 7.14. Diagrama E-R con un conjunto de entidades débiles.

En algunos casos, puede que el diseñador de la base de datos decida expresar un conjunto de entidades débiles como atributo compuesto multivalorado del conjunto de entidades propietarias. En el ejemplo, esta alternativa exigiría que el conjunto de entidades *asignatura* tuviera el atributo compuesto y multivalorado *sección*. Los conjuntos de entidades débiles se pueden modelar mejor como atributos si solo participan en la relación identificadora y tienen pocos atributos. A la inversa, las representaciones de los conjuntos de entidades débiles modelarán mejor las situaciones en las que esos conjuntos participen en otras relaciones, aparte de en la relación identificadora, y tengan muchos atributos. Queda claro que *sección* no cumple con los requisitos para ser modelado como un atributo compuesto multivalorado, y resulta más apropiado modelarlo como un conjunto de entidades débiles.

7.5.7. Diagrama E-R para la universidad

En la Figura 7.15 se muestra el diagrama E-R que se corresponde con el ejemplo de la universidad que se ha estado utilizando hasta ahora. Este diagrama E-R es equivalente a la descripción textual del modelo E-R de la universidad que se ha visto en la Sección 7.4, pero con algunas restricciones adicionales, y *sección* ahora como entidad débil.

En la base de datos de la universidad se tiene la restricción de que cada profesor ha de estar asociado exactamente a un departamento. Por ello, existe una línea doble en la Figura 7.15 entre *profesor* y *profesor_dept*, indicando la participación total de *profesor* en *profesor_dept*; es decir, todos los profesores tienen que estar asociados a un departamento. Más aún, existe una flecha desde *profesor_dept* a *departamento*, lo que indica que todos los profesores pueden tener, a lo sumo, un departamento asociado.

De forma similar, los conjuntos de entidades *asignatura* y *estudiante* tienen líneas dobles con los conjuntos de relaciones *asignatura_dept* y *estudiante_dept*, respectivamente, así como el conjunto de entidades *sección* con el conjunto de relaciones *secc_franja_horaria*. Las primeras dos relaciones, de hecho, tiene una línea dirigida apuntando a la otra relación, *departamento*, mientras que la tercera relación tiene un línea dirigida apuntando a *franja_horaria*.

Más aún, en la Figura 7.15 se muestra que el conjunto de relaciones *matricula* tiene un atributo descriptivo *nota*, y que todos los estudiantes tienen al menos un tutor. En la figura también se muestra que *sección* es un conjunto de entidades débiles, con atributos *secc_id*, *semestre* y *año* que forman el discriminador; *secc_asignatura* es el conjunto de relaciones identificadoras que relacionan el conjunto de entidades débiles *sección* con el conjunto de entidades fuertes *asignatura*.

En la Sección 7.6 se verá cómo se puede usar este diagrama E-R para derivar los distintos esquemas de relación que usamos.

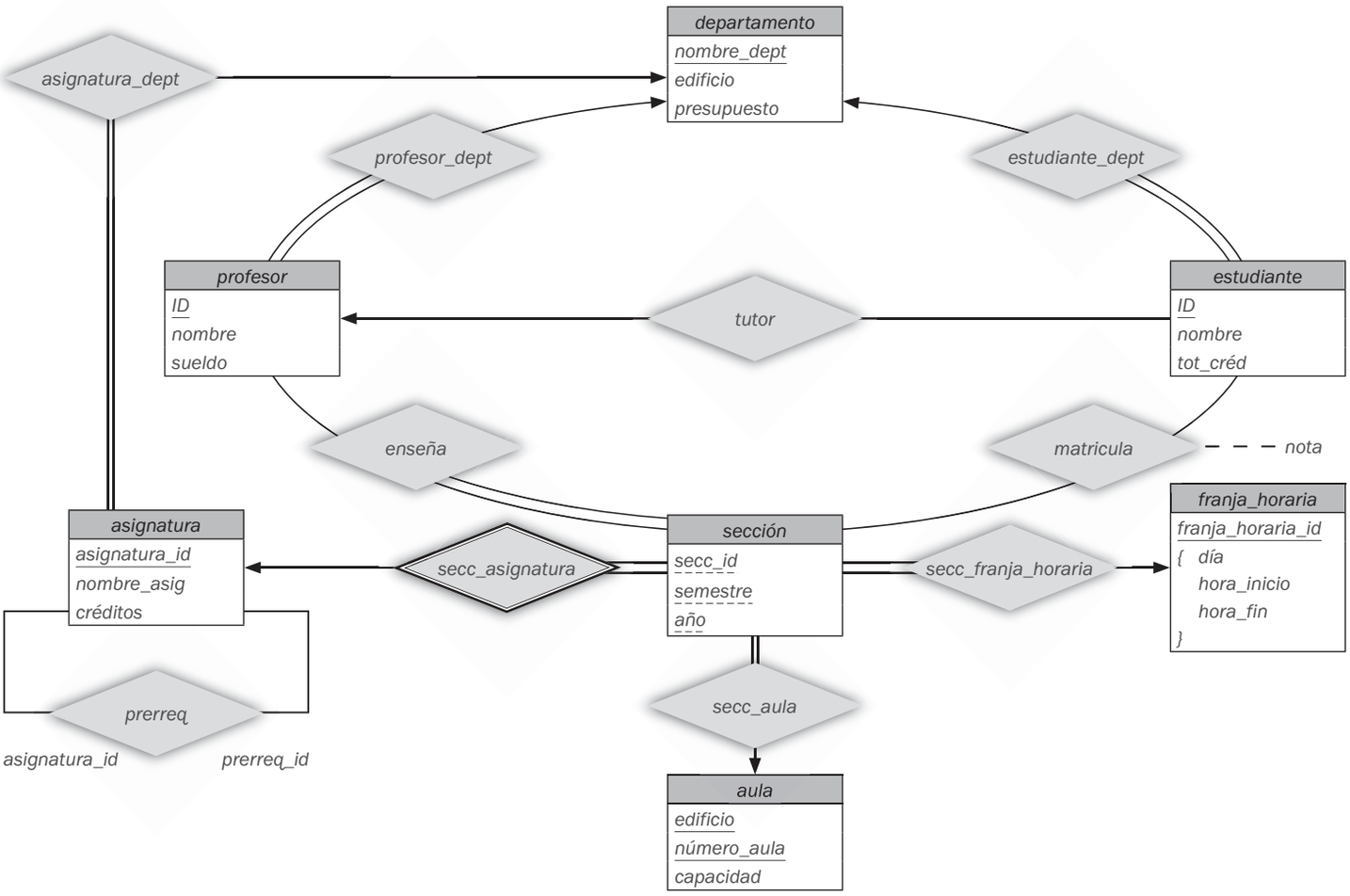


Figura 7.15. Diagrama E-R para una universidad.

7.6. Reducción a esquemas relacionales

Las bases de datos que se ajustan a un esquema de bases de datos E-R se pueden representar mediante conjuntos de esquemas de relación. Para cada conjunto de entidades y para cada conjunto de relaciones de la base de datos hay un solo esquema de relación a la que se asigna el nombre del conjunto de entidades o del conjunto de relaciones correspondiente.

Tanto el modelo E-R de bases de datos como el relacional son representaciones abstractas y lógicas de empresas del mundo real. Como los dos modelos usan principios de diseño parecidos, los diseños E-R se pueden convertir en diseños relacionales.

En esta sección se describe la manera de representar los esquemas E-R mediante esquemas de relación y el modo de asignar las restricciones que surgen del modelo E-R a restricciones de los esquemas de relación.

7.6.1. Representación de los conjuntos de entidades fuertes con atributos simples

Sea E un conjunto de entidades fuertes con los atributos descriptivos $a_1, a_2 \ldots, a_n$. Esta entidad se representa mediante un esquema denominado E con n atributos distintos. Cada tupla de las relaciones de este esquema corresponde a una entidad del conjunto de entidades E.

Para los esquemas derivados de los conjuntos de entidades fuertes, la clave primaria del conjunto de entidades sirve de clave primaria de los esquemas resultantes. Esto se deduce directamente del hecho de que cada tupla corresponde a una entidad concreta del conjunto de entidades.

Como ejemplo, considere el conjunto de entidades *préstamo* del diagrama E-R de la Figura 7.15. Este conjunto de entidades tiene tres atributos: *ID*, *nombre* y *tot_cred*. Este conjunto de entidades se representa mediante un esquema denominado *estudiante*, con tres atributos:

estudiante (ID, nombre, tot_créd)

Observe que, como *ID* es la clave primaria del conjunto de entidades, también es la clave primaria del esquema de la relación.

Siguiendo con el ejemplo, para el diagrama E-R de la Figura 7.15 todos los conjuntos de entidades fuertes, excepto *franja_horaria*, tienen solo atributos simples. Los esquemas que se derivan de estos conjuntos de entidades fuertes son:

aula (edificio, número_aula, capacidad)

departamento (nombre_dept, edificio, presupuesto)

asignatura (asignatura_id, nombre_asig, créditos)

profesor (ID, nombre, sueldo)

estudiante (ID, nombre, tot_créd)

Como puede observar, tanto el esquema *profesor* como el esquema *estudiante* son diferentes de los esquemas que se han usado en los capítulos anteriores (no contienen el atributo *nombre_dept*). Este aspecto se tratará en breve.

7.6.2. Representación de los conjuntos de entidades fuertes con atributos complejos

Cuando un conjunto de entidades fuertes tiene un conjunto de atributos que no son simples, las cosas son un poco más complicadas. Hemos tratado los atributos compuestos creando atributos separados para cada uno de los atributos componentes; no creamos un atributo separado para la composición de sus atributos compuestos. Para mostrar esto, considere la versión del conjunto de entidades *profesor* de la Figura 7.11. Para el atributo compuesto *nombre*, el esquema generado para *profesor* contiene los atributos *nombre*, *primer_apellido* y *segundo_apellido*; no existen atributos separados en el esquema para *nombre*. De forma similar, para el atributo compuesto *dirección* el esquema generado contiene los atributos *calle*, *ciudad*, *provincia* y *código postal*. Como *calle* es un atributo compuesto, se sustituye por *calle_nombre*, *calle_número* y *piso*. Se verá en la Sección 8.2.

Los atributos multivalorados se tratan de forma diferente al resto de atributos. Ya se ha visto que los atributos de un diagrama E-R se corresponden directamente con los atributos de los esquemas de relación apropiados. Sin embargo, los atributos multivalorados son una excepción; se crean nuevos esquemas de relación para ellos, como se verá en breve.

Los atributos derivados no se representan explícitamente en el modelo de datos relacional. Sin embargo, se representan como «métodos» en otros modelos de datos, como el modelo de datos objeto-relacional, que se describirá en el Capítulo 22.

El esquema relacional derivado de la versión del conjunto de entidades *profesor* con atributos compuestos, sin incluir el atributo multivalorado, es por tanto:

profesor (ID, nombre, primer_apellido, segundo_apellido, calle_nombre, calle_número, piso, ciudad, provincia, código_postal, fecha_de_nacimiento)

Para un atributo multivalorado M, se crea un esquema de relación R con un atributo A que corresponde a M y a los atributos correspondientes a la clave primaria del conjunto de entidades o de relaciones del que M es atributo.

Como ejemplo, considere el diagrama E-R de la Figura 7.11, donde se muestra el conjunto de entidades *profesor* con el atributo multivalorado *número_teléfono*. La clave primaria de *profesor* es *ID*. Para este atributo multivalorado se crea el esquema de relación

profesor_teléfono (ID, número_teléfono)

Cada número de teléfono de un profesor se representa como una única tupla de la relación de este esquema. Por tanto, si existe un profesor con *ID* 22222 y números de teléfono 655-1234 y 655-4321, la relación *profesor_teléfono* debería contener las dos tuplas (22222, 555-1234) y (22222, 555-4321).

Se crea una clave primaria del esquema de la relación consistente en todos los atributos del esquema. En el ejemplo anterior, la clave primaria consiste en todos los atributos de la relación *profesor_teléfono*.

Además, se crea una restricción de clave externa para el esquema de la relación, a partir del atributo multivalorado generado por efecto de la clave primaria del conjunto de entidades que hace referencia a la relación generada desde el conjunto de entidades. En el ejemplo anterior, la restricción de clave externa en la relación *profesor_teléfono* sería que el atributo *ID* hiciera referencia a la relación *profesor*.

En el caso de que el conjunto de entidades constara solo de dos atributos, un atributo de clave primaria B y un único atributo multivalorado M, el esquema de relación para el conjunto de entidades debería contener solamente un atributo, el atributo de clave primaria B. Se puede eliminar esta relación manteniendo el esquema de relación con el atributo B y el atributo A que corresponda con M.

Como ejemplo, considere el conjunto de entidades *franja_horaria* que se muestra en la Figura 7.15. Aquí, *franja_horaria_id* es la clave primaria del conjunto de entidades *franja_horaria* y existe un único atributo multivalorado que también resulta que es compuesto. El conjunto de entidades se puede representar por el siguiente esquema creado a partir del atributo compuesto multivalorado:

franja_horaria(franja_horaria_id, día, hora_inicio, hora_fin)

Aunque no se represente como una restricción en el diagrama E-R, se sabe que no puede haber dos convocatorias de clases que empiecen a la misma hora el mismo día de la semana, pero terminen a horas diferentes; según esta restricción, *hora_fin* se ha eliminado de la clave primaria del esquema *franja_horaria*.

La relación creada a partir del conjunto de entidades solo tendría un único atributo *franja_horaria_id*; la optimización al eliminar esta relación tiene la ventaja de la simplificación del esquema de la base de datos resultante, aunque tiene la desventaja relativa a las claves externas, que se tratará brevemente en la Sección 7.6.4.

7.6.3. Representación de los conjuntos de entidades débiles

Sea A un conjunto de entidades débiles con los atributos $a_1, a_2 \ldots, a_m$. Sea B el conjunto de entidades fuertes del que A depende. La clave primaria de B consiste en los atributos $b_1, b_2 \ldots, b_n$. El conjunto de entidades A se representa mediante el esquema de relación denominado A con un atributo por cada miembro del conjunto:

$$\{a_1, a_2, \ldots, a_m\} \cup \{b_1, b_2, \ldots, b_n\}$$

Para los esquemas derivados de conjuntos de entidades débiles la combinación de la clave primaria del conjunto de entidades fuertes y del discriminador del conjunto de entidades débiles sirve de clave primaria del esquema. Además de crear una clave primaria, también se crea una restricción de clave externa para la relación A, que especifica que los atributos $b_1, b_2 \ldots, b_n$ hacen referencia a la clave primaria de la relación B. La restricción de clave externa garantiza que por cada tupla que representa a una entidad débil existe la tupla correspondiente que representa a la entidad fuerte correspondiente.

Como ejemplo, considere el conjunto de entidades débiles *sección* del diagrama E-R de la Figura 7.15. Este conjunto de entidades tiene los atributos: *secc_id*, *semestre* y *año*. La clave primaria del conjunto de entidades *sección*, de la que depende *sección*, es *asignatura_id*. Por tanto, *sección* se representa mediante un esquema con los siguientes atributos:

sección(*asignatura_id*, *secc_id*, *semestre*, *año*)

La clave primaria consiste en la clave primaria del conjunto de entidades *asignatura* y el discriminador de *sección*, que es *secc_id*, *semestre* y *año*. También se crea una restricción de clave externa para el esquema *sección*, con el atributo *asignatura_id*, que hace referencia a la clave primaria del esquema *asignatura*, y la restricción de integridad «on delete cascade».[7] Por la especificación «on delete cascade» sobre la restricción de clave externa, si una entidad de asignatura se borra, entonces también se borran las entidades *sección* asociadas.

7.6.4. Representación de conjuntos de relaciones

Sea R un conjunto de relaciones, sea $a_1, a_2 \ldots, a_m$ el conjunto de atributos formado por la unión de las claves primarias de cada uno de los conjuntos de entidades que participan en R, y sean $b_1, b_2 \ldots, b_n$ los atributos descriptivos de R (si los hay). El conjunto de relaciones se representa mediante el esquema de relación R, con un atributo por cada uno de los miembros del conjunto:

$$\{a_1, a_2 \ldots, a_m\} \cup \{b_1, b_2 \ldots, b_n\}$$

Ya se ha descrito, en la Sección 7.3.3, la manera de escoger la clave primaria de un conjunto de relaciones binarias. Como se vio en ese apartado, tomar todos los atributos de las claves primarias de todos los conjuntos de entidades primarias sirve para identificar una tupla concreta pero, para los conjuntos de relaciones uno a uno, varios a uno y uno a varios, da como resultado un conjunto de atributos mayor del que hace falta para la clave primaria. En su lugar, la clave primaria se escoge de la siguiente forma:

- Para las relaciones binarias varios a varios la unión de los atributos de clave primaria de los conjuntos de entidades participantes pasa a ser la clave primaria.
- Para los conjuntos de relaciones binarias uno a uno la clave primaria de cualquiera de los conjuntos de entidades puede escogerse como clave primaria de la relación. La elección puede realizarse de manera arbitraria.
- Para los conjuntos de relaciones binarias varios a uno o uno a varios la clave primaria del conjunto de entidades de la parte «varios» de la relación sirve de clave primaria.
- Para los conjuntos de relaciones n-arias sin flechas en las líneas la unión de los atributos de clave primaria de los conjuntos de entidades participantes pasa a ser la clave primaria.
- Para los conjuntos de relaciones n-arias con una flecha en una de las líneas, las claves primarias de los conjuntos de entidades que no están en el lado «flecha» del conjunto de relaciones sirven de clave primaria del esquema. Recuerde que solo se permite una flecha saliente por cada conjunto de relaciones.

También se crean restricciones de clave externa para el esquema de relación R de la siguiente forma: para cada conjunto de entidades E_i relacionado con el conjunto de relaciones R se crea una restricción de clave externa de la relación R, con los atributos de R que se derivan de los atributos de clave primaria de E_i que hacen referencia a la clave primaria del esquema de relación que representa E_i.

Como ejemplo, considere el conjunto de relaciones *tutor* del diagrama E-R de la Figura 7.15. Este conjunto de relaciones implica a los dos conjuntos de entidades siguientes:

- *profesor* con clave primaria *ID*.
- *estudiante* con clave primaria *ID*.

Como el conjunto de relaciones no tiene ningún atributo, el esquema *tutor* tiene dos atributos, las claves primarias de *profesor* y *estudiante*. Como el conjunto de relaciones de *tutor* es del tipo varios a uno, se renombran a *p_ID* y *e_ID*.

También creamos dos restricciones de clave externa para la relación *tutor*, con el atributo *p_ID* referenciando a la clave primaria de *profesor* y el atributo *e_ID* referenciando a la clave primaria de *estudiante*.

Continuando con el ejemplo, para el diagrama E-R de la Figura 7.15, los esquemas derivados del conjunto de relaciones se muestran en la Figura 7.16.

Observe que para el caso de los conjuntos de relaciones *prerreq*, los indicadores de rol asociados con las relaciones se usan como nombre de atributo, ya que ambos roles se refieren a la misma relación *asignatura*.

De forma similar al caso de *tutor*, la clave primaria de cada una de las relaciones *secc_asignatura*, *secc_franja_horaria*, *secc_aula*, *profesor_dept*, *estudiante_dept* y *asignatura_dept* consta de las claves primarias de uno solo de los dos conjuntos de entidades relacionadas, ya que las relaciones correspondientes son del tipo varios a uno.

Las claves externas no se muestran en la Figura 7.16, pero para cada relación de la figura hay dos restricciones de clave externa, referenciando las dos relaciones creadas de los dos conjuntos de entidades relacionados. Por ejemplo, *secc_asignatura* tiene claves externas que remiten a *sección* y *aula*, *enseña* tiene claves externas que hacen referencia a *profesor* y *sección*, y *matricula* tiene claves externas que refieren a *estudiante* y *sección*.

7 La función «on delete cascade» de las restricciones de clave externa en SQL se describen en la Sección 4.4.5.

La optimización que nos ha permitido crear solo un único esquema de relación del conjunto de entidades *franja_horaria*, que es un atributo multivalorado, evita la creación de claves externas desde el esquema de relación *secc_franja_horaria* a la relación creada desde el conjunto de entidades *franja_horaria*, ya que se elimina la relación creada desde el conjunto de entidades *franja_horaria*. Se mantiene la relación creada desde el atributo multivalorado, y se nombra como *franja_horaria*, pero esta relación puede potencialmente no tener tuplas correspondientes a una *franja_horaria_id* o puede tener múltiples tuplas correspondientes a una *franja_horaria_id*, por lo que *franja_horaria_id* en *secc_franja_horaria* no puede referenciar esta relación.

El lector avispado puede preguntarse por qué hemos visto los esquemas *secc_asignatura*, *secc_franja_horaria*, *secc_aula*, *profesor_dept*, *estudiante_dept* y *asignatura_dept* en capítulos anteriores. La razón es que el algoritmo que se ha presentado hasta aquí genera algunos esquemas que se pueden eliminar o combinarse con otros. Estos aspectos se tratarán a continuación.

enseña (ID, asignatura_id, secc_id, semestre, año)

matricula (ID, asignatura_id, secc_id, semestre, año, nota)

prerreq (asignatura_id, prerreq_id)

tutor (e_ID, p_ID)

secc_asignatura (asignatura_id, secc_id, semestre, año)

secc_franja_horaria (asignatura_id, secc_id, semestre, año, franja_horaria_id)

secc_aula (asignatura_id, secc_id, semestre, año, edificio, número_aula)

profesor_dept (ID, nombre_dept)

estudiante_dept (ID, nombre_dept)

asignatura_dept (asignatura_id, nombre_dept)

Figura 7.16. Esquemas derivados del conjunto de relaciones del diagrama E-R de la Figura 7.15.

7.6.4.1. Redundancia de esquemas

Los conjuntos de relaciones que enlazan los conjuntos de entidades débiles con el conjunto correspondiente de entidades fuertes se tratan de manera especial. Como se hizo notar en la Sección 7.5.6, estas relaciones son del tipo varios a uno y no tienen atributos descriptivos. Además, la clave primaria de los conjuntos de entidades débiles incluye la clave primaria de los conjuntos de entidades fuertes. En el diagrama E-R de la Figura 7.14, el conjunto de entidades débiles *sección* depende del conjunto de entidades fuertes *asignatura* a través del conjunto de relaciones *secc_asignatura*. La clave primaria de *sección* es {*asignatura_id*, *secc_id*, *semestre*, *año*} y la clave primaria de *asignatura* es *asignatura_id*. Como *secc_asignatura* no tiene atributos descriptivos, el esquema *secc_asignatura* tiene los atributos, *asignatura_id*, *secc_id*, *semestre* y *año*. El esquema del conjunto de entidades *sección* tiene los atributos *asignatura_id*, *secc_id*, *semestre* y *año* (entre otros). Cada combinación (*asignatura_id*, *secc_id*, *semestre*, *año*) de una relación de *secc_asignatura* también se halla presente en el esquema de relación *sección*, y viceversa. Por tanto, el esquema *secc_asignatura* es redundante.

En general, el esquema de los conjuntos de relaciones que enlazan los conjuntos de entidades débiles con su conjunto correspondiente de entidades fuertes es redundante y no hace falta que esté presente en el diseño de la base de datos relacional basado en el diagrama E-R.

7.6.4.2. Combinación de esquemas

Considérese un conjunto *AB* de relaciones varios a uno del conjunto de entidades *A* al conjunto de entidades *B*. Usando el esquema de construcción de esquemas de relación descrito previamente se consiguen tres esquemas: *A*, *B* y *AB*. Entonces, los esquemas *A* y *B* se pueden combinar para formar un solo esquema consistente en la unión de los atributos de los dos esquemas. La clave primaria del esquema combinado es la clave primaria del conjunto de entidades en cuyo esquema se combinó el conjunto de relaciones.

Como ejemplo, examinemos las distintas relaciones del diagrama E-R de la Figura 7.15, que satisfacen los criterios anteriores:

- *profesor_dept*. Los esquemas *profesor* y *departamento* corresponden a los conjuntos de entidades *A* y *B*, respectivamente. Entonces, el esquema *profesor_dept* se puede combinar con el esquema *profesor*. El esquema *profesor* resultado consta de los atributos {*ID*, *nombre*, *nombre_dept*, *sueldo*}.
- *estudiante_dept*. Los esquemas *estudiante* y *departamento* corresponden a los conjuntos de entidades *A* y *B*, respectivamente. Entonces, el esquema *estudiante_dept* se puede combinar con el esquema *estudiante*. El esquema *estudiante* resultado consta de los atributos {*ID*, *nombre*, *nombre_dept*, *tot_créd*}.
- *asignatura_dept*. Los esquemas *asignatura* y *departamento* corresponden a los conjuntos de entidades *A* y *B*, respectivamente. Entonces, el esquema *asignatura_dept* se puede combinar con el esquema *asignatura*. El esquema *asignatura* resultado consta de los atributos {*asignatura_id*, *nombre_asig*, *nombre_dept*, *créditos*}.
- *secc_aula*. Los esquemas *sección* y *aula* corresponden a los conjuntos de entidades *A* y *B*, respectivamente. Entonces, el esquema *secc_aula* se puede combinar con el esquema *sección*. El esquema *sección* resultado consta de los atributos {*asignatura_id*, *secc_id*, *semestre*, *año*, *edificio*, *número_aula*}.
- *secc_franja_horaria*. Los esquemas *sección* y *franja_horaria* corresponden a los conjuntos de entidades *A* y *B*, respectivamente. Entonces, el esquema *secc_franja_horaria* se puede combinar con el esquema *sección* obtenido en el paso previo. El esquema *sección* resultado consta de los atributos {*asignatura_id*, *secc_id*, *semestre*, *año*, *edificio*, *número_aula*, *franja_horaria_id*}.

En el caso de las relaciones uno a uno, el esquema de relación del conjunto de relaciones puede combinarse con el esquema de cualquiera de los conjuntos de entidades.

Se pueden combinar esquemas aunque la participación sea parcial, usando los valores nulos. En el ejemplo anterior, si *profesor_dept* fuese parcial se almacenarían valores nulos para el atributo *nombre_dept* de los profesores que no tuviesen asociado un departamento.

Por último, vamos a considerar las restricciones de clave externa que habrían aparecido en el esquema que representa a los conjuntos de relaciones. Habría habido restricciones de clave externa referenciando a cada uno de los conjuntos de entidades que participan en los conjuntos de relaciones. Se eliminan las restricciones referenciando los conjuntos de entidades en aquellos esquemas en que se combina el esquema del conjunto de relaciones, y se añade la otra restricción de clave externa al esquema combinado. Por ejemplo, *profesor_dept* tiene una restricción de clave externa del atributo *nombre_dept* referenciando la relación *departamento*. Esta restricción externa se añade a la relación *profesor* cuando el esquema *profesor_dept* se combina en *profesor*.

7.7. Aspectos del diseño entidad-relación

Los conceptos de conjunto de entidades y de conjunto de relaciones no son precisos, y es posible definir un conjunto de entidades y las relaciones entre ellas de diferentes formas. En esta sección se examinan aspectos básicos del diseño de esquemas de bases de datos E-R. La Sección 7.10 trata el proceso de diseño con más detalle.

7.7.1. Uso de conjuntos de entidades en lugar de atributos

Considérese el conjunto de entidades *profesor* con el atributo adicional *número_teléfono* (Figura 7.17a). Se puede argumentar que un *teléfono* es una entidad en sí misma con los atributos *número_teléfono* y *ubicación*; la ubicación puede ser la oficina o el domicilio en el que el teléfono está instalado, y se pueden representar los teléfonos móviles (celulares) mediante el valor «móvil». Si se adopta este punto de vista, no añadimos el atributo *número_teléfono* a *profesor*. En su lugar se crea:

- El conjunto de entidades *teléfono* con los atributos *número_teléfono* y *ubicación*.
- Un conjunto de relaciones *profesor_teléfono*, que denota la asociación entre los profesores y sus teléfonos.

  Esta alternativa es la que se muestra en la Figura 7.17b.

¿Cuál es entonces la diferencia principal entre estas dos definiciones de profesor? Tratar el teléfono como un atributo *número_teléfono* implica que cada profesor tiene exactamente un número de teléfono. Tratar el teléfono como una entidad *teléfono* permite que los empleados tengan varios números de teléfono (incluyendo ninguno) asociados. Sin embargo, se puede definir fácilmente en su lugar *número_teléfono* como atributo multivalorado para permitir varios teléfonos por profesor.

La diferencia principal es que tratar el teléfono como entidad modela mejor una situación en la que se puede querer guardar información extra sobre el teléfono, como su ubicación, su tipo (móvil, teléfono IP o fijo) o las personas que lo comparten. Por tanto, tratar el teléfono como entidad es más general que tratarlo como atributo y resulta adecuado cuando la generalidad pueda ser útil.

En cambio, no sería apropiado tratar el atributo *nombre* (de un profesor) como entidad; es difícil argumentar que *nombre* sea una entidad en sí misma (a diferencia de lo que ocurre con *teléfono*). Así pues, resulta adecuado tener *nombre* como atributo del conjunto de entidades *profesor*.

Por tanto, se suscitan dos preguntas: ¿qué constituye un atributo? y ¿qué constituye un conjunto de entidades? Desafortunadamente no hay respuestas sencillas. Las distinciones dependen principalmente de la estructura de la empresa real que se esté modelando y de la semántica asociada con el atributo en cuestión.

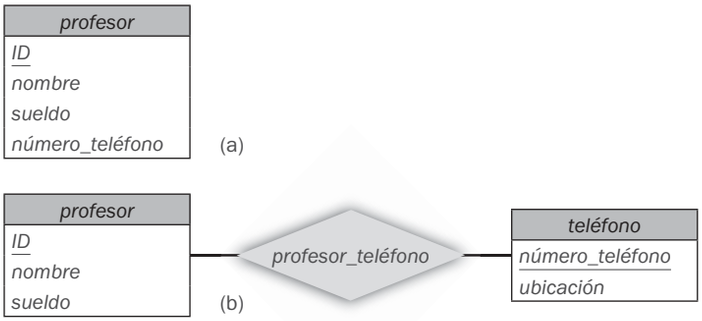


Figura 7.17. Alternativas para añadir *teléfono* al conjunto de entidades *profesor*.

Un error común es usar la clave primaria de un conjunto de entidades como atributo de otro conjunto de entidades en lugar de usar una relación. Por ejemplo, es incorrecto modelar el *ID* de un estudiante como atributo de *profesor*, aunque cada profesor solo tutele a un estudiante. La relación *tutor* es la forma correcta de representar la conexión entre estudiantes y profesores, ya que hace explícita su conexión, en lugar de dejarla implícita mediante un atributo.

Otro error relacionado con este que se comete a veces es escoger los atributos de clave primaria de los conjuntos de entidades relacionados como atributos del conjunto de relaciones. Por ejemplo, *ID* (el atributo de clave primaria de *estudiante*) e *ID* (la clave primaria de *profesor*) no deben aparecer como atributos de la relación *tutor*. Esto no es adecuado, ya que los atributos de clave primaria están implícitos en el conjunto de relaciones.[8]

7.7.2. Uso de conjuntos de entidades en lugar de conjuntos de relaciones

No siempre está claro si es mejor expresar un objeto mediante un conjunto de entidades o mediante un conjunto de relaciones. En la Figura 7.15 se usa el conjunto de relaciones *matricula* para modelar la situación en la que un estudiante se matricula de una (sección de) asignatura. Una alternativa es imaginar que existe un registro de los registro-asignatura de cada curso en que se matricula un estudiante. Entonces tenemos un conjunto de entidades que representan los registros registro-asignatura. Llamemos a este conjunto de entidades *registro*. Las entidades *registro* están relacionadas con, exactamente, un estudiante y una sección, por lo que tenemos dos conjuntos de relaciones, uno que relaciona registro-asignatura con estudiantes y otro que relaciona registro-asignatura con secciones. En la Figura 7.18 se muestran los conjuntos de entidades *sección* y *estudiante* de la Figura 7.15 con los conjuntos de relaciones *matricula* sustituidos por un conjunto de entidades y dos conjuntos de relaciones:

- *registro*, el conjunto de entidades que representan los rubros de registro-asignatura.
- *sección_reg*, el conjunto de relaciones que relacionan *registro* y *asignatura*.
- *estudiante_reg*, el conjunto de relaciones que relacionan *registro* y *estudiante*.

Se han utilizado líneas dobles para indicar la participación total de las entidades *registro*.

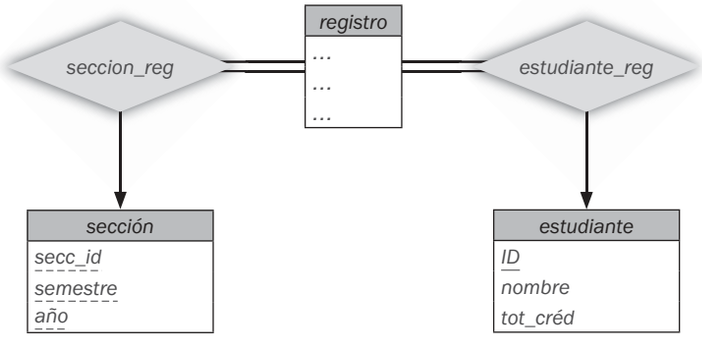


Figura 7.18. Sustitución de *matricula* por registro y dos conjuntos de relaciones.

8 Cuando se crea un esquema de relación a partir del esquema E-R, los atributos pueden aparecer en un esquema generado desde el conjunto de relaciones *tutor*, como se verá más adelante; no obstante, no deben aparecer en el conjunto de relaciones *tutor*.

Tanto la forma de la Figura 7.15 como la Figura 7.18 representan con precisión la información de la universidad, pero el uso de *matricula* es más compacto y probablemente preferible. Sin embargo, si la oficina de matriculación asocia otra información con un registro de matricula, podría ser mejor hacer de ello una entidad propia.

Un criterio para determinar si se debe usar un conjunto de entidades o un conjunto de relaciones puede ser escoger un conjunto de relaciones para describir las acciones que se produzcan entre entidades. Este enfoque también puede ser útil para decidir si ciertos atributos se pueden expresar mejor como relaciones.

7.7.3. Conjuntos de relaciones binarias y *n*-arias

Las relaciones en las bases de datos suelen ser binarias. Puede que algunas relaciones que no parecen ser binarias se puedan representar mejor mediante varias relaciones binarias. Por ejemplo, se puede crear la relación ternaria *padres*, que relaciona a cada hijo con su padre y con su madre. Sin embargo, esa relación se puede representar mediante dos relaciones binarias, *padre* y *madre*, que relacionan a cada hijo con su padre y con su madre por separado. El uso de las dos relaciones *padre* y *madre* permite el registro de la madre del niño aunque no se conozca la identidad del padre; si se usara en la relación ternaria *padres* se necesitaría un valor nulo. En este caso es preferible usar conjuntos de relaciones binarias.

De hecho, siempre es posible sustituir los conjuntos de relaciones no binarias (n-aria, para $n > 2$) por varios conjuntos de relaciones binarias. Por simplificar, considere el conjunto de relaciones abstracto ternario ($n = 3$) R y los conjuntos de entidades A, B y C. Se sustituye el conjunto de relaciones R por un conjunto de entidades E y se crean tres conjuntos de relaciones como se muestra en la Figura 7.19:

- R_A, que relaciona E y A.
- R_B, que relaciona E y B.
- R_C, que relaciona E y C.

Si el conjunto de relaciones R tiene atributos, estos se asignan al conjunto de entidades E; además, se crea un atributo de identificación especial para E (ya que se deben poder distinguir las diferentes entidades de cada conjunto de entidades con base en los valores de sus atributos). Para cada relación (a_i, b_i, c_i) del conjunto de relaciones R se crea una nueva entidad e_i del conjunto de entidades E. Luego, en cada uno de los tres nuevos conjuntos de relaciones, se inserta una relación del modo siguiente:

- (e_i, a_i) en R_A.
- (e_i, b_i) en R_B.
- (e_i, c_i) en R_C.

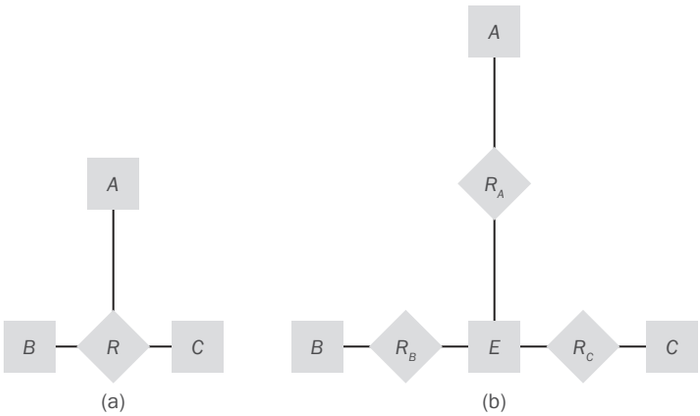


Figura 7.19. Relaciones ternarias frente a tres relaciones binarias.

Este proceso se puede generalizar de forma directa a los conjuntos de relaciones n-arias. Por tanto, conceptualmente, se puede restringir el modelo E-R para que solo incluya conjuntos de relaciones binarias. Sin embargo, esta restricción no siempre es deseable.

- Es posible que sea necesario crear un atributo de identificación para que el conjunto de entidades represente el conjunto de relaciones. Este atributo, junto con los conjuntos de relaciones adicionales necesarios, incrementa la complejidad del diseño y (como se verá en la Sección 7.6) los requisitos globales de almacenamiento.
- Un conjunto de relaciones n-arias muestra más claramente que varias entidades participan en una sola relación.
- Puede que no haya forma de traducir las restricciones a la relación ternaria en restricciones a las relaciones binarias. Por ejemplo, considérese una restricción que dice que R es del tipo varios a uno de A, B a C; es decir, cada par de entidades de A y de B se asocia, a lo sumo, con una entidad de C. Esta restricción no se puede expresar mediante restricciones de cardinalidad sobre los conjuntos de relaciones R_A, R_B, y R_C.

Considere el conjunto de relaciones *proy_direc* de la Sección 7.2.2, que relaciona *profesor*, *estudiante* y *proyecto*. No se puede dividir directamente *proy_direc* en relaciones binarias entre *profesor* y *proyecto* y entre *profesor* y *estudiante*. Si se hiciese, se podría registrar que el profesor Katz trabaja en los proyectos A y B con los estudiantes Shankar y Zhang; sin embargo, no se podría registrar que Katz trabaja en el proyecto A con el estudiante Shankar y trabaja en el proyecto B con el estudiante Zhang, pero no trabaja en el proyecto A con Zhang ni en el proyecto B con Shankar.

El conjunto de relaciones *proy_direc* se puede dividir en relaciones binarias mediante la creación de nuevos conjuntos de entidades como se ha descrito anteriormente. Sin embargo, no sería muy natural.

7.7.4. Ubicación de los atributos de las relaciones

La razón de cardinalidad de una relación puede afectar a la ubicación de sus atributos de las relaciones. Por tanto, los atributos de los conjuntos de relaciones uno a uno o uno a varios pueden estar asociados con uno de los conjuntos de entidades participantes, en lugar de con el conjunto de relaciones. Por ejemplo, especifiquemos que *tutor* es un conjunto de relaciones uno a varios tal que cada profesor puede tutelar a varios estudiantes, pero cada estudiante solo puede tener como tutor a un único profesor. En este caso, el atributo *fecha*, que especifica la fecha en que el profesor se convirtió en tutor del estudiante, podría estar asociado con el conjunto de entidades *estudiante*, como se muestra en la Figura 7.20 (para simplificar la figura solo se muestran algunos de los atributos de los dos conjuntos de entidades). Dado que cada entidad *estudiante* participa en una relación con un ejemplar de *profesor*, como máximo, hacer esta designación de atributos tendría el mismo significado que colocar *fecha* en el conjunto de relaciones *tutor*. Los atributos de un conjunto de relaciones uno a varios solo se pueden recolocar en el conjunto de entidades de la parte «varios» de la relación. Para los conjuntos de entidades uno a uno, los atributos de la relación se pueden asociar con cualquiera de las entidades participantes.

La decisión de diseño sobre la ubicación de los atributos descriptivos en estos casos —como atributo de la relación o de la entidad— debe reflejar las características de la empresa que se modela. El diseñador puede elegir mantener *fecha* como atributo de *tutor* para expresar explícitamente que la fecha se refiere a la relación de tutela y no a otros aspectos del estatus del estudiante en la universidad (por ejemplo, la fecha de aceptación en la universidad).

La elección de la ubicación de atributos es más sencilla para los conjuntos de relaciones varios a varios. Volviendo al ejemplo, espe-

cifiquemos el caso, quizá más realista, de que *tutor* sea un conjunto de relaciones varios a varios que expresa que un profesor puede tutelar a uno o más estudiantes y que un estudiante puede tener como tutor a uno o más profesores. Si hay que expresar la fecha en que un profesor se convirtió en tutor de un determinado estudiante, *fecha* debe ser atributo del conjunto de relaciones *tutor*, en lugar de serlo de cualquiera de las entidades participantes. Si *fecha* fuese un atributo de *estudiante*, por ejemplo, no se podría determinar qué profesor se convirtió en tutor en dicha fecha. Cuando un atributo se determina mediante la combinación de los conjuntos de entidades participantes, en lugar de por cada entidad por separado, ese atributo debe estar asociado con el conjunto de relaciones varios a varios. La Figura 7.3 muestra la ubicación de *fecha* como atributo de la relación; de nuevo, para simplificar la figura solo se muestran algunos de los atributos de los dos conjuntos de entidades.

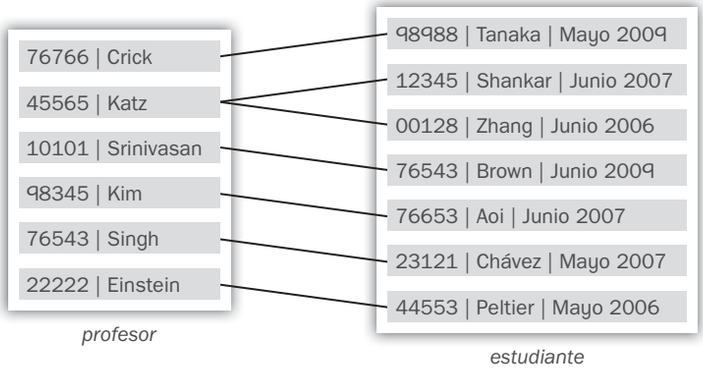


Figura 7.20. *fecha* como un atributo del conjunto de entidades *estudiante*.

7.8. Características del modelo E-R extendido

Aunque los conceptos básicos del modelo E-R pueden modelar la mayor parte de las características de las bases de datos, algunos aspectos de estas se pueden expresar mejor mediante ciertas extensiones del modelo E-R básico. En este apartado se estudian las características E-R extendidas de especialización, generalización, conjuntos de entidades de nivel superior e inferior, herencia de atributos y agregación.

Para facilitar el contenido se usará un esquema de base de datos algo más elaborado. En particular, se modelarán las distintas personas de la universidad definiendo un conjunto de entidades *persona*, con atributos *ID*, *nombre* y *dirección*.

7.8.1. Especialización

Los conjuntos de entidades pueden incluir subgrupos de entidades que se diferencian de alguna forma de las demás entidades del conjunto. Por ejemplo, un subconjunto de entidades de un conjunto de entidades puede tener atributos que no sean compartidos por todas las entidades del conjunto de entidades. El modelo E-R ofrece un medio para representar estos grupos de entidades diferentes.

Como ejemplo, el conjunto de entidades *persona* se puede clasificar como uno de los siguientes:

- *empleado*.
- *estudiante*.

Cada uno de estos tipos de persona se describe mediante un conjunto de atributos que incluye todos los atributos del conjunto de entidades *persona* más otros posibles atributos adicionales. Por ejemplo, las entidades *empleado* se pueden describir además mediante el atributo *sueldo*, mientras que las entidades *estudiante* se pueden describir además mediante el atributo *tot_créd*. El proceso de establecimiento de subgrupos dentro del conjunto de entidades se denomina **especialización.** La especialización de *persona* permite distinguir entre las personas basándonos en si son empleados o estudiantes: en general, cada persona puede ser empleado, estudiante, las dos cosas o ninguna de ellas.

Como ejemplo adicional, supóngase que la universidad desea dividir a los estudiantes en dos categorías: graduados y pregraduados. Los estudiantes graduados tienen una oficina asignada. Los estudiantes pregraduados tienen asignada una residencia. Cada uno de estos tipos de estudiantes se describe mediante un conjunto de atributos que incluye todos los atributos del conjunto de entidades *estudiante* más otros atributos adicionales.

La universidad puede crear dos especializaciones de *estudiante*, por ejemplo, *graduado* y *pregraduado*. Como ya se ha visto, las entidades *estudiante* se describen mediante los atributos *ID*, *nombre*, *dirección* y *tot_créd*. El conjunto de entidades *graduado* tendría todos los atributos de *estudiante* y el atributo adicional *número_oficina*. El conjunto de entidades *pregraduado* tendría todos los atributos de *estudiante* y el atributo adicional *residencia*.

La especialización se puede aplicar repetidamente para refinar el esquema de diseño. Por ejemplo, los empleados de la universidad se pueden clasificar como uno de los siguientes:

- *profesor*.
- *secretaria*.

Cada uno de estos tipos de empleado se describe mediante un conjunto de atributos que incluye todos los atributos del conjunto de entidades *empleado* y otros adicionales. Por ejemplo, las entidades *profesor* se pueden describir, además, por el atributo *rango*; mientras que las entidades *secretaria* se describen por el atributo *horas_semana*. Además, las entidades *secretaria* pueden participar en la relación *secretaria_para*, entre las entidades *secretaria* y *empleado*, que identifica a los empleados a los que ayuda una secretaria.

Un conjunto de entidades se puede especializar en más de una característica distintiva. En este ejemplo, la característica distintiva entre las entidades *empleado* es el trabajo que desempeña cada empleado. Otra especialización coexistente se puede basar en si es un trabajador temporal o fijo, lo que da lugar a los conjuntos de entidades *empleado_temporal* y *empleado_fijo*. Cuando se forma más de una especialización en un conjunto de entidades, cada entidad concreta puede pertenecer a varias especializaciones. Por ejemplo, un empleado dado puede ser un empleado temporal y a su vez secretaria.

En términos de los diagramas E-R, la especialización se representa mediante una punta de flecha hueca desde la entidad especializada a la otra entidad (véase la Figura 7.21). Esta relación se denomina relación ES (ISA – «is a», es un/una) y representa, por ejemplo, que un profesor «es» un/una empleado/a.

La forma en que se dibuja la especialización en un diagrama E-R depende de si una entidad puede pertenecer a varios conjuntos de entidades especializados o si debe pertenecer, como mucho, a un conjunto de entidades especializado. En el primer caso (se permiten varios conjuntos) se llama **especialización solapada,** mientras que en el segundo (se permite uno como mucho) se denomina **especialización disjunta.** Para una especialización solapada (como ocurre en el caso de *estudiante* y *empleado* como especialización de *persona*), se usan dos flechas separadas. Para una especialización disjunta (como en el caso de *profesor* y *secretaria* como especialización de *empleado*), se usa una única flecha. La relación de especialización también se puede llamar relación **superclase-subclase.** Los conjuntos de entidades de mayor y menor nivel son conjuntos de entidades normales, es decir, rectángulos que contienen el nombre del conjunto de entidades.

7.8.2. Generalización

El refinamiento a partir del conjunto de entidades inicial en sucesivos niveles de subgrupos de entidades representa un proceso de diseño **descendente (top-down)** en el que las distinciones se hacen explícitas. El proceso de diseño también puede proceder de forma **ascendente (bottom-up),** en la que varios conjuntos de entidades se sintetizan en un conjunto de entidades de nivel superior basado en características comunes. El diseñador de la base de datos puede haber identificado primero:

- El conjunto de entidades *profesor*, con los atributos *profesor_id*, *profesor_nombre*, *profesor_sueldo* y *rango*.
- El conjunto de entidades *secretaria*, con los atributos *secretaria_id*, *secretaria_nombre*, *secretaria_sueldo* y *horas_semana*.

Existen similitudes entre el conjunto de entidades *profesor* y el conjunto de entidades *secretaria* en el sentido de que tienen varios atributos que, conceptualmente, son iguales en los dos conjuntos de entidades: los atributos para el identificador, el nombre y el sueldo. Esta similitud se puede expresar mediante la **generalización,** que es una relación de contención que existe entre el conjunto de entidades de *nivel superior* y uno o varios conjuntos de entidades de *nivel inferior*. En este ejemplo, *empleado* es el conjunto de entidades de nivel superior y *profesor* y *secretaria* son conjuntos de entidades de nivel inferior. En este caso, los atributos que son conceptualmente iguales tienen nombres diferentes en los dos conjuntos de entidades de nivel inferior. Para crear generalizaciones, los atributos deben tener un nombre común y representarse mediante la entidad de nivel superior *persona*. Se pueden usar los nombres de atributos *ID*, *nombre*, *dirección*, como se vio en el ejemplo de la Sección 7.8.1.

Los conjuntos de entidades de nivel superior e inferior también se pueden denominar con los términos **superclase** y **subclase,** respectivamente. El conjunto de entidades *persona* es la superclase de las subclases *estudiante* y *empleado*.

A efectos prácticos, la generalización es una inversión simple de la especialización. Se aplicarán ambos procesos, combinados, en el transcurso del diseño del esquema E-R de una empresa. En términos del propio diagrama E-R no se distingue entre especialización y generalización. Los niveles nuevos de representación de las entidades se distinguen (especialización) o sintetizan (generalización) cuando el esquema de diseño llega a expresar completamente la aplicación de la base de datos y los requisitos del usuario de la base de datos. Las diferencias entre los dos enfoques se pueden caracterizar mediante su punto de partida y su objetivo global.

La especialización parte de un único conjunto de entidades; destaca las diferencias entre las entidades del conjunto mediante la creación de diferentes conjuntos de entidades de nivel inferior. Esos conjuntos de entidades de nivel inferior pueden tener atributos o participar en relaciones que no se aplican a todas las entidades del conjunto de entidades de nivel superior. Realmente, la razón de que el diseñador aplique la especialización es poder representar esas características distintivas. Si *estudiante* y *empleado* tuvieran exactamente los mismos atributos que las entidades *persona* y participaran en las mismas relaciones en las que participan las entidades *persona*, no habría necesidad de especializar el conjunto de entidades *persona*.

La generalización parte del reconocimiento de que varios conjuntos de entidades comparten algunas características comunes (es decir, se describen mediante los mismos atributos y participan en los mismos conjuntos de relaciones). Con base en esas similitudes, la generalización sintetiza esos conjuntos de entidades en un solo conjunto de nivel superior. La generalización se usa para destacar las similitudes entre los conjuntos de entidades de nivel inferior y para ocultar las diferencias; también permite una economía de representación, ya que no se repiten los atributos compartidos.

7.8.3. Herencia de los atributos

Una propiedad crucial de las entidades de nivel superior e inferior creadas mediante la especialización y la generalización es la **herencia de los atributos.** Se dice que los atributos de los conjuntos de entidades de nivel superior son **heredados** por los conjuntos de entidades de nivel inferior. Por ejemplo, *estudiante* y *empleado* heredan los atributos de *persona*. Así, *estudiante* se describe mediante sus atributos *ID*, *nombre* y *dirección* y, adicionalmente, por el atributo *tot_créditos*; *empleado* se describe mediante sus atributos *ID*, *nombre* y *dirección* y, adicionalmente, por el atributo *salario*. La herencia de los atributos se aplica en todos los niveles de los conjuntos de entidades de nivel inferior; así, *profesor* y *secretaria*, que son subclases de *empleado*, heredan los atributos *ID*, *nombre* y *dirección* de *persona*, además de heredar *sueldo* de *empleado*.

Los conjuntos de entidades de nivel inferior (o subclases) también heredan la participación en los conjuntos de relaciones en los que participa su entidad de nivel superior (o superclase). Al igual que con la herencia de los atributos, la herencia de participación también se aplica en todos los niveles de los conjuntos de entidades de nivel inferior. Por ejemplo, suponga que el conjunto de entidades *persona* participa en una relación *persona_dept* con *departamento*. Entonces, los conjuntos de entidades *estudiante*, *empleado*, *profesor* y *secretaria*, que son subclases de los conjuntos de entidades *persona*, también participan en la relación *persona_dept* con *departamento*. Los conjuntos de entidades anteriores pueden participar en cualquier relación en la que participe el conjunto de entidades *persona*.

Tanto si se llega a una porción dada del modelo E-R mediante la especialización como si se hace mediante la generalización, el resultado es básicamente el mismo:

- Un conjunto de entidades de nivel superior con los atributos y las relaciones que se aplican a todos sus conjuntos de entidades de nivel inferior.
- Conjuntos de entidades de nivel inferior con características distintivas que solo se aplican en un conjunto dado de entidades de nivel inferior.

En lo que sigue, aunque a menudo solo se haga referencia a la generalización, las propiedades que se estudian corresponden completamente a ambos procesos.

La Figura 7.21 describe una **jerarquía** de conjuntos de entidades. En la figura, *empleado* es un conjunto de entidades de nivel inferior de *persona* y un conjunto de entidades de nivel superior de los conjuntos de entidades *profesor* y *secretaria*. En las jerarquías, un conjunto de entidades dado solo puede estar implicado como conjunto de entidades de nivel inferior en una relación ES; es decir, los conjuntos de entidades de este diagrama solo tienen **herencia única.** Si un conjunto de entidades es un conjunto de entidades de nivel inferior en más de una relación ES, el conjunto de entidades tiene **herencia múltiple** y la estructura resultante se denomina *retículo*.

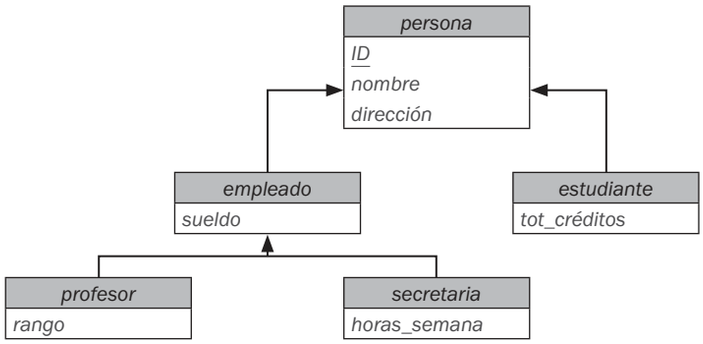


Figura 7.21. Especialización y generalización.

7.8.4. Restricciones a las generalizaciones

Para modelar una empresa con más precisión, el diseñador de la base de datos puede decidir imponer ciertas restricciones sobre una generalización concreta. Un tipo de restricción implica la determinación de las entidades que pueden formar parte de un conjunto de entidades de nivel inferior dado. Esa pertenencia puede ser una de las siguientes:

- **Definida por la condición.** En los conjuntos de entidades de nivel inferior definidos por la condición, la pertenencia se evalúa en función del cumplimiento de una condición o predicado explícito por la entidad. Por ejemplo, supóngase que el conjunto de entidades de nivel superior *estudiante* tiene el atributo *tipo_estudiante*. Todas las entidades *estudiante* se evalúan según el atributo *tipo_estudiante* que las define. Solo las entidades que satisfacen la condición *tipo_estudiante* = «graduado» pueden pertenecer al conjunto de entidades de nivel inferior *estudiante_graduado*. Todas las entidades que satisfacen la condición *tipo_estudiante* = «pregraduado» se incluyen en *estudiante_pregraduado*. Dado que todas las entidades de nivel inferior se evalúan en función del mismo atributo (en este caso, *tipo_estudiante*), se dice que este tipo de generalización está **definida por el atributo.**
- **Definida por el usuario.** Los conjuntos de entidades de nivel inferior definidos por el usuario no están restringidos por una condición de pertenencia; más bien, el usuario de la base de datos asigna las entidades a un conjunto de entidades dado. Por ejemplo, supóngase que, después de tres meses de trabajo, los empleados de la universidad se asignan a uno de los cuatro grupos de trabajo. En consecuencia, los grupos se representan como cuatro conjuntos de entidades de nivel inferior del conjunto de entidades de nivel superior *empleado*. No se asigna cada empleado a una entidad grupo concreta automáticamente de acuerdo con una condición explícita que lo defina. En vez de eso, la asignación al grupo la lleva a cabo el usuario que toma la decisión persona a persona. La asignación se implementa mediante una operación que añade cada entidad a un conjunto de entidades.

Un segundo tipo de restricciones tiene relación con la pertenencia de las entidades a más de un conjunto de entidades de nivel inferior de la generalización. Los conjuntos de entidades de nivel inferior pueden ser de uno de los tipos siguientes:

- **Disjuntos.** La *restricción sobre la condición de disjunción* exige que cada entidad no pertenezca a más de un conjunto de entidades de nivel inferior. En el ejemplo, la entidad *estudiante* solo puede cumplir una condición del atributo *tipo_estudiante*; una entidad puede ser un estudiante graduado o pregraduado, pero no ambas cosas a la vez.
- **Solapados.** En las *generalizaciones solapadas* la misma entidad puede pertenecer a más de un conjunto de entidades de nivel inferior de la generalización. Como ejemplo, considere el grupo de trabajo de empleados y suponga que algunos empleados participan en más de un grupo de trabajo. Cada empleado, por tanto, puede aparecer en más de uno de los conjuntos de entidades grupo, que son conjuntos de entidades de nivel inferior de *empleado*. Por tanto, la generalización es solapada.

En la Figura 7.21 supondremos que una persona puede ser a la vez un empleado y un estudiante. Se mostrará esta generalización solapada con flechas separadas: una de empleado a persona y otra desde estudiante a persona. Sin embargo, la generalización de un profesor y una secretaria es disjunta. Se mostrará utilizando flechas simples.

Una última restricción, la **restricción de completitud** sobre una generalización o especialización, especifica si una entidad del conjunto de entidades de nivel superior debe pertenecer, al menos, a uno de los conjuntos de entidades de nivel inferior de la generalización o especialización. Esta restricción puede ser de uno de los tipos siguientes:

- **Generalización** o **especialización total.** Cada entidad de nivel superior debe pertenecer a un conjunto de entidades de nivel inferior.
- **Generalización** o **especialización parcial.** Puede que alguna entidad de nivel superior no pertenezca a ningún conjunto de entidades de nivel inferior.

La generalización parcial es la predeterminada. Se puede especificar la generalización total en los diagramas E-R añadiendo la palabra «total» en el diagrama y dibujando una línea discontinua desde la palabra a la correspondiente cabeza de flecha hueca en la que aplica (para una generalización total), o al conjunto de cabezas de flecha huecas a las que aplica (para una generalización solapada).

La generalización de *estudiante* es total: todas las entidades *estudiante* deben ser graduados o pregraduados. Como el conjunto de entidades de nivel superior al que se llega mediante la generalización suele estar compuesto únicamente de entidades de los conjuntos de entidades de nivel inferior, la restricción de completitud para los conjuntos de entidades de nivel superior generalizados suele ser total. Cuando la generalización es parcial, las entidades de nivel superior no están limitadas a aparecer en los conjuntos de entidades de nivel inferior. Los conjuntos de entidades grupo de trabajo ilustran una especialización parcial. Como los empleados solo se asignan a cada grupo después de llevar tres meses en el trabajo, puede que algunas entidades *empleado* no pertenezcan a ninguno de los conjuntos de entidades grupo de nivel inferior.

Los conjuntos de entidades equipo se pueden caracterizar mejor como especialización de *empleado* parcial y solapada. La generalización de *estudiante_graduado* y *estudiante_pregraduado* en *estudiante* es una generalización total y disjunta. Las restricciones de completitud y sobre la condición de disjunción, sin embargo, no dependen una de la otra. Las características de las restricciones también pueden ser parcial — disjunta y total— y solapada.

Es evidente que algunos requisitos de inserción y de borrado son consecuencia de las restricciones que se aplican a una generalización o especialización dada. Por ejemplo, cuando se impone una restricción de completitud total, las entidades insertadas en un conjunto de entidades de nivel superior se deben insertar, al menos, en uno de los conjuntos de entidades de nivel inferior. Con una restricción de definición por condición, todas las entidades de nivel superior que cumplen la condición se deben insertar en ese conjunto de entidades de nivel inferior. Finalmente, las entidades que se borren de los conjuntos de entidades de nivel superior se deben borrar también de todos los conjuntos de entidades de nivel inferior asociados a los que pertenezcan.

7.8.5. Agregación

Una limitación del modelo E-R es que no es posible expresar relaciones entre las relaciones. Para ilustrar la necesidad de estos constructores, considérese la relación ternaria *proy_direc*, que ya se ha visto anteriormente, entre *profesor*, *estudiante* y *proyecto* (véase la Figura 7.13).

Suponga ahora que se desea que cada profesor que dirige a un estudiante en un proyecto realice un informe de evaluación mensual. Se modela el informe de evaluación como una entidad *evaluación*, con una clave primaria *evaluación_id*. Una alternativa para registrar la combinación (*estudiante*, *proyecto*, *profesor*) a la que corresponde una evaluación es crear un conjunto de relaciones cuaternaria *eval_para* entre *estudiante*, *proyecto*, *profesor* y *evaluación*. Se requiere una relación cuaternaria, una relación binaria entre *estudiante* y *evaluación*, por ejemplo, no permitiría repre-

sentar la combinación (*proyecto*, *profesor*) a la que corresponde la *evaluación*. Utilizando los constructores del modelo E-R básico se obtiene el diagrama de la Figura 7.22 (para simplificar se han omitido los atributos de los conjuntos de entidades).

Parece que los conjuntos de relaciones *proy_direc* y *eval_para* se pueden combinar en un solo conjunto de relaciones. No obstante, no se deben combinar en una sola relación, ya que puede que algunas combinaciones *estudiante*, *proyecto*, *profesor* no tengan una *evaluación* asociada.

No obstante, hay información redundante en la figura obtenida, ya que cada combinación *estudiante*, *proyecto*, *profesor* de *eval_para* también tiene que estar en *proy_direc*. Si *evaluación* fuese un valor en lugar de una entidad, se podría hacer que *evaluación* fuese un atributo multivalorado de la relación *proy_direc*. Sin embargo, esta alternativa no puede ser una opción si una *evaluación* puede estar relacionada con otras entidades; por ejemplo, los informes de evaluación pueden estar asociados con una *secretaria* que es la responsable de procesar el informe de evaluación para realizar los pagos escolares.

La mejor forma de modelar una situación como la descrita es usar la agregación. La **agregación** es una abstracción a través de la cual las relaciones se tratan como entidades de nivel superior. Así, para este ejemplo, se considera el conjunto de relaciones *proy_direc* (que relaciona los conjuntos de entidades *estudiante*, *proyecto*, *profesor*) como el conjunto de entidades de nivel superior denominado *proy_direc*. Ese conjunto de entidades se trata de la misma forma que cualquier otro conjunto de entidades. Se puede crear entonces la relación binaria *eval_para* entre *proy_direc* y *evaluación* para representar para qué combinación (*estudiante*, *proyecto*, *profesor*) es la evaluación. La Figura 7.23 muestra una notación para la agregación que se usa habitualmente para representar esta situación.

7.8.6. Reducción a esquemas relacionales

Ya estamos en disposición de describir cómo las características del modelo E-R extendido se pueden trasladar a esquemas relacionales.

7.8.6.1. Representación de la generalización

Existen dos métodos diferentes para designar los esquemas de relación de los diagramas E-R que incluyen generalización. Aunque en esta discusión se hace referencia a la generalización de la Figura 7.21, se simplifica incluyendo solo la primera capa de los conjuntos de entidades de nivel inferior, es decir, *empleado* y *estudiante*. Se da por supuesto que *ID* es la clave primaria de *persona*.

1. Se crea un esquema para el conjunto de entidades de nivel superior. Para cada conjunto de entidades de nivel inferior se crea un esquema que incluye un atributo para cada uno de los atributos de ese conjunto de entidades más un atributo por cada atributo de la clave primaria del conjunto de entidades de nivel superior. Así, para el diagrama E-R de la Figura 7.21, se tienen tres esquemas:

   persona (ID, nombre, calle, ciudad)

   empleado (ID, sueldo)

   estudiante (ID, tot_créd)

   Los atributos de clave primaria del conjunto de entidades de nivel superior pasan a ser atributos de clave primaria del conjunto de entidades de nivel superior y de todos los conjuntos de entidades de nivel inferior. En el ejemplo anterior se pueden ver subrayados.

   Además, se crean restricciones de clave externa para los conjuntos de entidades de nivel inferior, con sus atributos de clave primaria que hacen referencia a la clave primaria de la relación creada a partir del conjunto de entidades de nivel superior. En el ejemplo anterior, el atributo *ID* de *empleado* haría referencia a la clave primaria de *persona*, y similar para *estudiante*.
2. Es posible una representación alternativa si la generalización es disjunta y completa; es decir, si no hay ninguna entidad miembro de dos conjuntos de entidades de nivel inferior directamente por debajo de un conjunto de entidades de nivel superior, y si todas las entidades del conjunto de entidades de nivel superior también pertenecen a uno de los conjuntos de entidades de nivel inferior. En este caso no se crea un esquema para el conjunto de

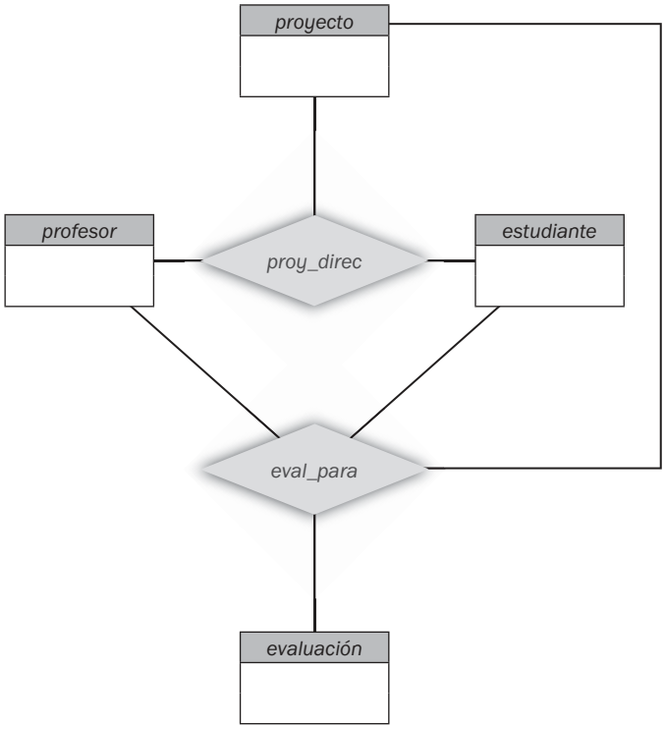


Figura 7.22. Diagrama E-R con relaciones redundantes.

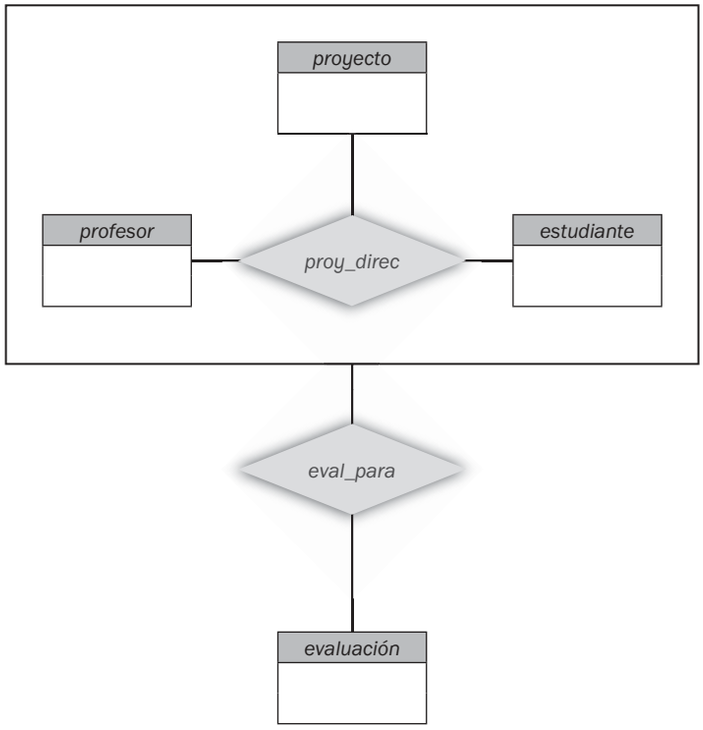


Figura 7.23. Diagrama E-R con agregación.

entidades de nivel superior. En vez de eso, para cada conjunto de entidades de nivel inferior se crea un esquema que incluye un atributo por cada atributo de ese conjunto de entidades más un atributo por *cada* atributo del conjunto de entidades de nivel superior. Entonces, para el diagrama E-R de la Figura 7.21, se tienen dos esquemas:

empleado (ID, nombre, calle, ciudad, sueldo)

estudiante (ID, nombre, calle, ciudad, tot_cred)

Estos dos esquemas tienen *ID*, que es el atributo de clave primaria del conjunto de entidades de nivel superior *persona*, como clave primaria.

Un inconveniente del segundo método es la definición de las restricciones de clave externa. Para ilustrar el problema, supóngase que se tiene un conjunto de relaciones *R* que implica al conjunto de entidades *persona*. Con el primer método, al crear un esquema de relación *R* a partir del conjunto de relaciones, también se define una restricción de clave externa para *R*, que hace referencia al esquema *persona*. Desafortunadamente, con el segundo método no se tiene una única relación a la que pueda hacer referencia la restricción de clave externa de *R*. Para evitar este problema hay que crear un esquema de relación *persona* que contenga, al menos, los atributos de clave primaria de la entidad *persona*.

Si se usara el segundo método para una generalización solapada, algunos valores se almacenarían varias veces de manera innecesaria. Por ejemplo, si una persona es a la vez empleado y estudiante, los valores de *calle* y de *ciudad* se almacenarían dos veces.

Si la generalización fuera disjunta pero no completa; es decir, si alguna persona no fuera ni empleado ni estudiante, entonces haría falta un esquema adicional

persona (ID, nombre, calle, ciudad)

para representar a esas personas. Sin embargo, aún permanecería el problema con las restricciones de clave externa indicados anteriormente. En un intento de soslayar el problema, suponga que a los empleados y a los estudiantes se les representa adicionalmente en la relación *persona*. Desafortunadamente, la información de nombre, calle y ciudad no debería guardarse de forma redundante en la relación *persona* y en la relación *estudiante* para los estudiantes, y de forma similar en la relación *persona* y en la relación *empleado* para los empleados. Esto sugiere guardar la información de nombre, calle y ciudad solo en la relación *persona* y eliminar esta información de *estudiante* y de *empleado*. Si lo hacemos, el resultado es exactamente el obtenido con el primer método presentado.

7.8.6.2. Representación de la agregación

El diseño de esquemas para los diagramas E-R que incluyen agregación es sencillo. Considérese el diagrama de la Figura 7.23. El esquema del conjunto de relaciones *eval_para* entre la agregación de *proy_direc* y el conjunto de entidades *evaluación* incluye un atributo para cada atributo de las claves primarias del conjunto de entidades *evaluación* y del conjunto de relaciones *proy_direc*. También incluye un atributo para los atributos descriptivos, si los hay, del conjunto de relaciones *eval_para*. Entonces, se transforman los conjuntos de relaciones y de entidades de la entidad agregada siguiendo las reglas que se han definido anteriormente.

Las reglas que se han visto anteriormente para la creación de restricciones de clave primaria y de clave externa para los conjuntos de relaciones, se pueden aplicar también a los conjuntos de relaciones que incluyen agregación, tratando la agregación como cualquier otra entidad. La clave primaria de la agregación es la clave primaria del conjunto de relaciones que la define. No hace falta ninguna relación más para que represente la agregación; en vez de eso, se usa la relación creada a partir de la relación definidora.

7.9. Notaciones alternativas para el modelado de datos

Una representación con un diagrama del modelo de datos de una aplicación es una parte importante del diseño del esquema de la base de datos. La creación de un esquema de la base de datos requiere no solo expertos en el modelado de datos, sino también expertos en el dominio que conozcan los requisitos de la aplicación pero que puede que no estén familiarizados con el modelado de datos. Que una representación gráfica sea intuitiva es importante, ya que facilita la comunicación de información entre estos dos grupos de expertos.

Se han propuesto distintas notaciones alternativas para el modelado de datos, siendo los diagramas E-R y los diagramas de clases de UML los más utilizados. No existe un estándar universal para la notación de los diagramas E-R y en cada libro se utilizan notaciones diferentes. Se ha elegido una notación en particular para esta sexta edición que difiere de la notación empleada en ediciones anteriores, por razones que se explicarán más adelante en esta sección.

En el resto de la sección se estudiarán algunas de las notaciones gráficas E-R alternativas, así como la notación de los diagramas de clases de UML. Para ayudar en la comparación de nuestra notación con las alternativas, la Figura 7.24 resume el conjunto de símbolos que se ha usado en los diagramas E-R.

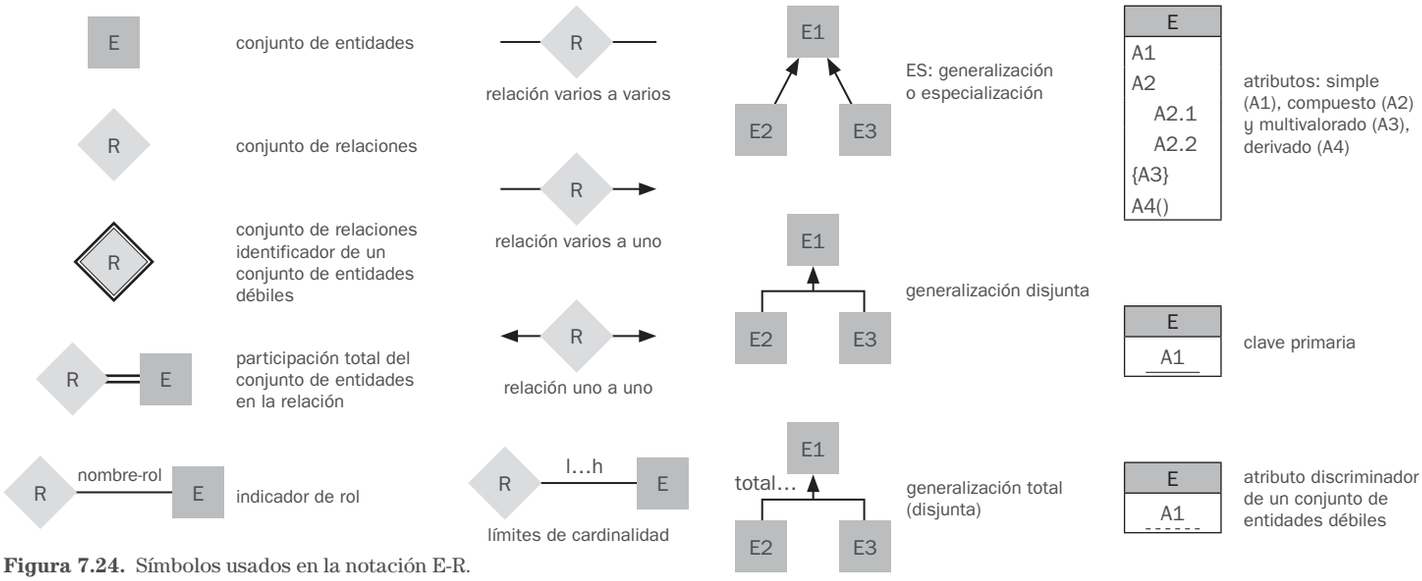


Figura 7.24. Símbolos usados en la notación E-R.

7.9.1. Notaciones E-R alternativas

La Figura 7.25 indica algunas de las notaciones de diagramas E-R alternativas que se usan habitualmente. Una representación alternativa de los atributos o entidades es mostrarlos en óvalos conectados al rectángulo que representa la entidad; los atributos de claves primarias se indican subrayándolos. La notación anterior se muestra en la parte superior de la figura. Los atributos de las relaciones se pueden representar de forma parecida, conectando los óvalos al rombo que representa la relación.

Las restricciones de cardinalidad en las relaciones se pueden indicar de varias formas diferentes, como se muestra en la Figura 7.25. En una alternativa que se muestra en la parte izquierda de la figura, las etiquetas $*$ y 1 en los segmentos que salen de las relaciones se usan a menudo para denotar relaciones varios a varios, uno a uno y varios a uno. El caso de uno a varios es simétrico con el de varios a uno, y no se muestra.

En otra notación alternativa que se muestra en la parte derecha de la figura, los conjuntos de relaciones se representan mediante líneas entre los conjuntos de entidades, sin rombos; por tanto, solo se podrán modelar relaciones binarias. Las restricciones de cardinalidad en esta notación se muestran mediante la notación «pata de gallo», como en la figura. En una relación *R* entre *E1* y *E2* la «pata de gallo» en ambos lados indica una relación varios a varios, mientras que una «pata de gallo» solo en el lado de *E1* indica una relación varios a uno desde *E1* a *E2*. La participación total se especifica en esta notación mediante una barra vertical. Fíjese, sin embargo, que en una relación *R* entre entidades *E1* y *E2* si la participación de *E1* en *R* es total, la barra vertical se sitúa en el lado opuesto, adyacente a la entidad *E2*. De forma similar, la participación parcial se indica usando un círculo, de nuevo en el lado opuesto.

En la parte inferior de la Figura 7.25 se muestra una representación alternativa de la generalización, usando triángulos en lugar de cabezas de flecha huecas.

En anteriores ediciones de este libro, hasta la quinta edición, se han usado óvalos para representar los atributos, con triángulos para representar la generalización, como se muestra en la Figura 7.25. La notación utilizando óvalos para los atributos y rombos para las relaciones es similar a la forma original de los diagramas E-R usados por Chen en su artículo que introdujo la notación del modelado E-R. Esta notación se conoce actualmente como notación de Chen.

El Instituto Nacional de Estados Unidos para Normalización y Tecnología ha definido en 1993 un estándar llamado IDEFIX. IDEFIX usa la notación «pata de gallo» con barras verticales al lado de las relaciones para denotar la participación total y círculos huecos para denotar la participación parcial, e incluye otras notaciones que no se han incluido.

Con el crecimiento del uso del lenguaje de marcado unificado (UML: *Unified Markup Language*), que se describe en la Sección 7.9.2, se ha decidido actualizar la notación E-R para acercarla al formato de los diagramas de clases de UML; las conexiones se verán más claras en la Sección 7.9.2. En comparación con nuestra notación previa, la nueva notación proporciona una representación más compacta de los atributos y está más cercana a las notaciones que admiten muchas de las herramientas de modelado E-R, además de ser más cercana a la notación de los diagramas de clases de UML.

Existen varias herramientas para construir diagramas E-R, cada una de ellas con sus propias variantes de notación. Algunas de las herramientas proporcionan la posibilidad de elegir entre varias notaciones E-R. Para más información consulte las referencias en las notas bibliográficas.

Una de las diferencias clave entre los conjuntos de entidades en un diagrama E-R y los esquemas de relación creados desde dichas entidades es que los atributos en el esquema relacional que se corresponden a relaciones E-R, como el atributo *nombre_dept* de *profesor*, no se muestran en el conjunto de entidades del diagrama E-R. Algunas herramientas de modelado de datos permiten que los usuarios elijan entre vistas de la misma entidad, una vista sin dichos atributos y otra vista relacional con dichos atributos.

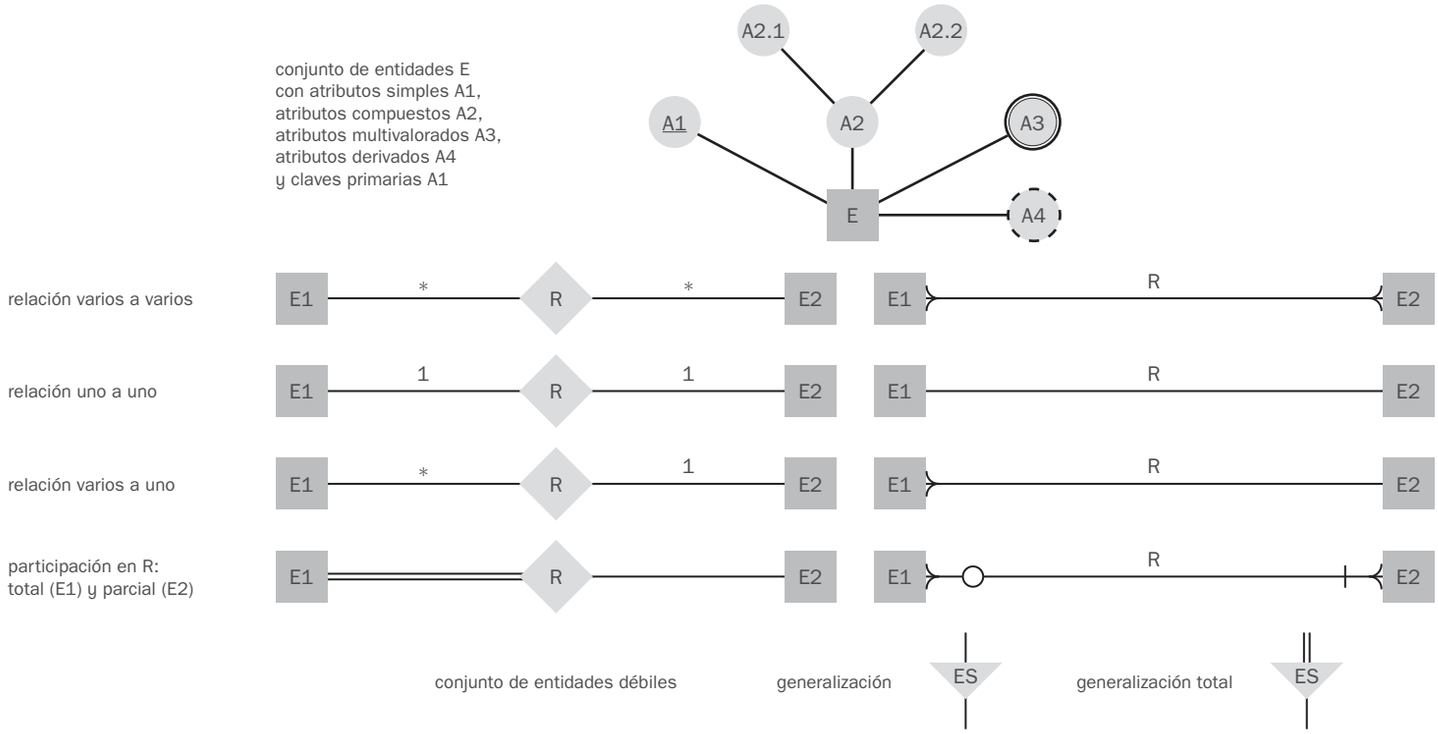


Figura 7.25. Notaciones E-R alternativas.

7.9.2. El lenguaje de modelado unificado UML

Los diagramas entidad-relación ayudan a modelar el componente de representación de datos de los sistemas de software. La representación de datos, sin embargo, solo forma parte del diseño global del sistema. Otros componentes son los modelos de interacción del usuario con el sistema, la especificación de los módulos funcionales del sistema y su interacción, etc. El **lenguaje de modelado unificado** (Unified Modeling Language, UML) es una norma desarrollada bajo los auspicios del Grupo de Administración de Objetos (Object Management Group, OMG) para la creación de especificaciones de diferentes componentes de los sistemas de software. Algunas de las partes del UML son:

- **Diagramas de clase.** Los diagramas de clase son parecidos a los diagramas E-R. Más adelante en esta sección se mostrarán algunas características de los diagramas de clase y del modo en que se relacionan con los diagramas E-R.
- **Diagramas de caso de uso.** Los diagramas de caso de uso muestran la interacción entre los usuarios y el sistema, en especial los pasos de las tareas que llevan a cabo los usuarios (como retirar dinero o matricularse en una asignatura).
- **Diagramas de actividad.** Los diagramas de actividad describen el flujo de tareas entre los diferentes componentes del sistema.
- **Diagramas de implementación.** Los diagramas de implementación muestran los componentes del sistema y sus interconexiones, tanto en el nivel de los componentes de software como en el de hardware.

Aquí no se pretende ofrecer un tratamiento detallado de las diferentes partes del UML. Véanse las notas bibliográficas para encontrar referencias sobre el UML. En vez de eso, se mostrarán algunas características de la parte del UML que se relaciona con el modelado de datos mediante ejemplos.

La Figura 7.26 muestra varios constructores de diagramas E-R y sus constructores equivalentes de diagramas de clases de UML. Más adelante se describen estos constructores. UML modela objetos, mientras que E-R modela entidades. Los objetos son como entidades y tienen atributos, pero también proporcionan un conjunto de funciones (denominadas métodos) que se pueden invocar para calcular valores con base en los atributos de los objetos, o para actualizar el propio objeto. Los diagramas de clases pueden describir métodos, además de atributos. Los objetos se tratan en el Capítulo 22. UML no admite atributos compuestos ni multivalorados, y los atributos derivados son equivalentes a los métodos sin parámetros. Como las clases admiten encapsulación, UML permite que los atributos y los métodos tengan el prefijo «+», «−» o «#», que significan acceso público, privado o protegido, respectivamente. Los atributos privados solo los pueden usar los métodos de la clase, mientras que los atributos protegidos solo los pueden usar los métodos de la clase y sus subclases, lo que debería resultar familiar a cualquiera que conozca Java, C++ o C#.

En terminología de UML, los conjuntos de relaciones se conocen como **asociaciones;** nos referiremos a ellas como conjuntos de relaciones por consistencia con la terminología E-R. Los conjuntos de relaciones binarias se representan en UML dibujando simplemente una línea que conecte los conjuntos de entidades. El nombre del conjunto de relaciones se escribe junto a la línea. También se puede especificar el rol que desempeña cada conjunto de entidades en un conjunto de relaciones escribiendo el nombre del rol sobre la línea, junto al conjunto de entidades. De manera alternativa, se puede escribir el nombre del conjunto de relaciones en un recuadro, junto con los atributos del conjunto de relaciones, y conectar el recuadro con una línea discontinua a la línea que describe el conjunto de relaciones. Este recuadro se puede tratar entonces como un conjunto de entidades, de la misma forma que la agregación en los diagramas E-R, y puede participar en relaciones con otros conjuntos de entidades.

Desde la versión 1.3 de UML, UML soporta las relaciones no binarias usando la misma notación de rombos utilizada en los diagramas E-R. En versiones anteriores de UML las relaciones no binarias no se podían representar directamente, había que convertirlas en relaciones binarias usando la técnica que se vio en la Sección 7.7.3. UML soporta que se use la notación rombo incluso para relaciones binarias, pero la mayoría de los diseñadores utilizan líneas.

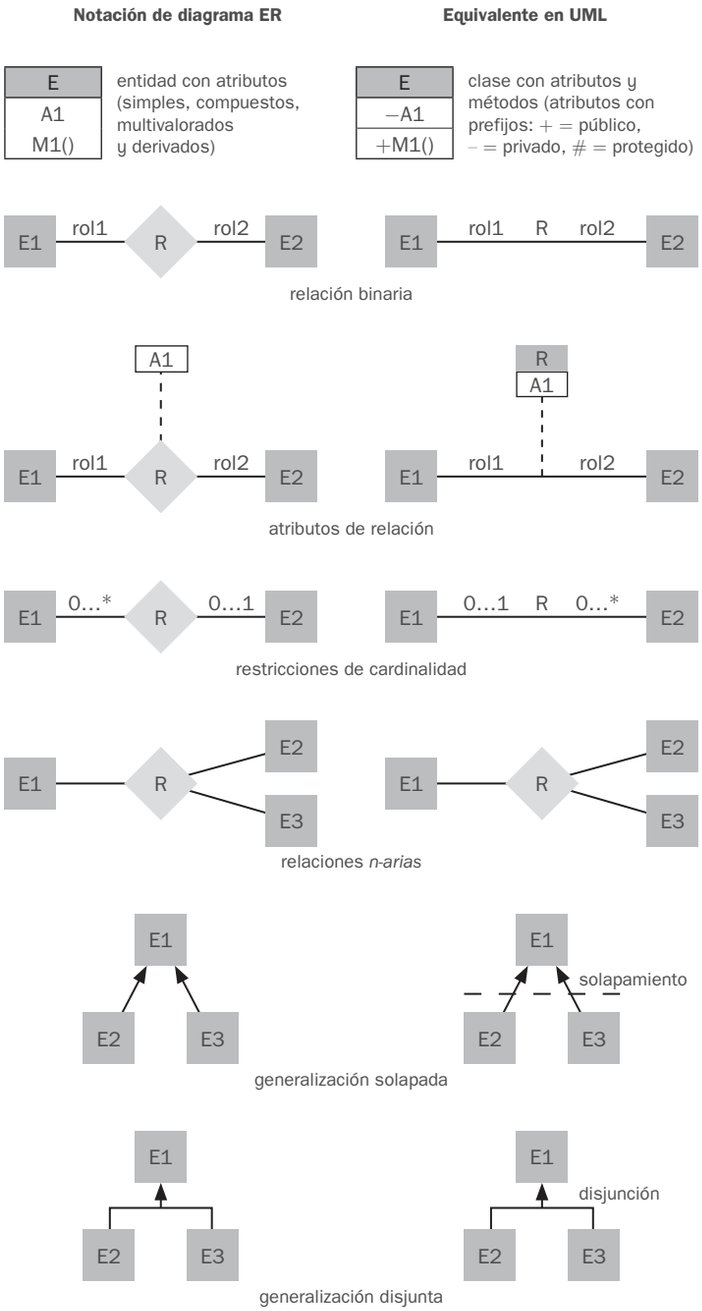


Figura 7.26. Símbolos usados en la notación de diagramas de clase UML.

Las restricciones de cardinalidad se especifican en UML de la misma forma que en los diagramas E-R, de la forma $l\ldots h$, donde l denota el número mínimo y h el máximo de relaciones en que puede participar cada entidad. Sin embargo, hay que ser consciente de que la ubicación de las restricciones es exactamente la contraria que en los diagramas E-R, como se muestra en la Figura 7.26. La restricción $0\ldots*$ en el lado $E2$ y $0\ldots1$ en el lado $E1$ significa que cada entidad $E2$ puede participar, a lo sumo, en una relación, mientras que cada entidad $E1$ puede participar en varias relaciones; en otras palabras, la relación es varios a uno de $E2$ a $E1$.

Los valores aislados, como 1 o $*$, se pueden escribir en los arcos; el valor 1 sobre un arco se trata como equivalente de $1\ldots1$, mientras que $*$ es equivalente a $0\ldots*$. UML soporta la generalización; la notación es básicamente la misma que en la notación E-R, incluyendo la representación de generalizaciones disjuntas y solapadas.

Los diagramas de clases de UML incluyen otras notaciones que no se corresponden con las notaciones E-R vistas hasta ahora. Por ejemplo, una línea entre dos conjuntos de entidades con un rombo en un extremo especifica que la entidad en el extremo del rombo contiene a la otra (la inclusión se denomina «agregación» en la terminología de UML; no confundir este uso de agregación con el sentido en que se usa en el modelo E-R). Por ejemplo, una entidad vehículo puede contener una entidad motor.

Los diagramas de clases de UML también ofrecen notaciones para representar características de los lenguajes orientados a objetos, como las interfaces. Véanse las referencias en las notas bibliográficas para obtener más información sobre los diagramas de clases de UML.

7.10. Otros aspectos del diseño de bases de datos

La explicación sobre el diseño de esquemas dada en este capítulo puede crear la falsa impresión de que el diseño de esquemas es el único componente del diseño de bases de datos. En realidad, hay otras consideraciones que se tratarán con más profundidad en capítulos posteriores y que se describirán brevemente a continuación.

7.10.1. Restricciones de datos y diseño de bases de datos relacionales

Se ha visto gran variedad de restricciones de datos que pueden expresarse mediante SQL, como las restricciones de clave primaria, las de clave externa, las restricciones **check,** los asertos y los disparadores. Las restricciones tienen varios propósitos. El más evidente es la automatización de la conservación de la consistencia. Al expresar las restricciones en el lenguaje de definición de datos de SQL, el diseñador puede garantizar que el propio sistema de bases de datos haga que se cumplan las restricciones. Esto es más digno de confianza que dejar que cada programa haga cumplir las restricciones por su cuenta. También ofrece una ubicación central para la actualización de las restricciones y la adición de otras nuevas.

Otra ventaja de definir explícitamente las restricciones es que algunas resultan especialmente útiles en el diseño de esquemas de bases de datos relacionales. Si se sabe, por ejemplo, que el número de DNI identifica de manera unívoca a cada persona, se puede usar el número de DNI de una persona para vincular los datos relacionados con esa persona aunque aparezcan en varias relaciones. Compare esta posibilidad, por ejemplo, con el color de ojos, que no es un identificador unívoco. El color de ojos no se puede usar para vincular los datos correspondientes a una persona concreta en varias relaciones, ya que los datos de esa persona no se podrían distinguir de los datos de otras personas con el mismo color de ojos.

En la Sección 7.6 se generó un conjunto de esquemas de relación para un diseño E-R dado mediante las restricciones especificadas en el diseño. En el Capítulo 8 se formaliza esta idea y otras relacionadas y se muestra la manera en que puede ayudar al diseño de esquemas de bases de datos relacionales. El enfoque formal del diseño de bases de datos relacionales permite definir de manera precisa la bondad de cada diseño y mejorar los diseños malos. Se verá que el proceso de comenzar con un diseño entidad-relación y generar mediante algoritmos los esquemas de relación a partir de ese diseño es una buena manera de comenzar el proceso de diseño.

Las restricciones de datos también resultan útiles para determinar la estructura física de los datos. Puede resultar útil almacenar físicamente próximos en el disco los datos que están estrechamente relacionados entre sí, de modo que se mejore la eficiencia del acceso al disco. Algunas estructuras de índices funcionan mejor cuando el índice se crea sobre una clave primaria.

La aplicación de las restricciones se lleva a cabo a un precio potencialmente alto en rendimiento cada vez que se actualiza la base de datos. En cada actualización el sistema debe comprobar todas las restricciones y rechazar las actualizaciones que no las cumplen o ejecutar los disparadores correspondientes. La importancia de la penalización en rendimiento no solo depende de la frecuencia de actualización, sino también del modo en que se haya diseñado la base de datos. En realidad, la eficiencia de la comprobación de determinados tipos de restricciones es un aspecto importante de la discusión del diseño de esquemas para bases de datos relacionales que se verá en el Capítulo 8.

7.10.2. Requisitos de uso: consultas y rendimiento

El rendimiento de los sistemas de bases de datos es un aspecto crítico de la mayor parte de los sistemas informáticos empresariales. El rendimiento no solo tiene que ver con el uso eficiente del hardware de cálculo y de almacenamiento que se usa, sino también con la eficiencia de las personas que interactúan con el sistema y de los procesos que dependen de los datos de las bases de datos.

Existen dos métricas principales para el rendimiento:

- **Productividad:** el número de consultas o actualizaciones (a menudo denominadas *transacciones*) que pueden procesarse en promedio por unidad de tiempo.
- **Tiempo de respuesta:** el tiempo que tarda *una sola* transacción desde el comienzo hasta el final en promedio o en el peor de los casos.

Los sistemas que procesan gran número de transacciones agrupadas por lotes se centran en tener una productividad elevada. Los sistemas que interactúan con personas y los sistemas de tiempo crítico suelen centrarse en el tiempo de respuesta. Estas dos métricas no son equivalentes. La productividad elevada se consigue mediante un elevado uso de los componentes del sistema. Ello puede dar lugar a que algunas transacciones se pospongan hasta el momento en que puedan ejecutarse con mayor eficiencia. Las transacciones pospuestas sufren un bajo tiempo de respuesta.

Históricamente, la mayor parte de los sistemas de bases de datos comerciales se han centrado en la productividad; no obstante, gran variedad de aplicaciones, incluidas las aplicaciones basadas en la web y los sistemas informáticos para telecomunicaciones necesitan un buen tiempo de respuesta promedio y una cota razonable para el peor tiempo de respuesta que pueden ofrecer.

La comprensión de los tipos de consultas que se espera que sean más frecuentes ayuda al proceso de diseño. Las consultas que implican reuniones necesitan evaluar más recursos que las que no las implican. A veces, cuando se necesita una reunión, puede que el administrador de la base de datos decida crear un índice que facilite la evaluación de la reunión. Para las consultas —tanto si está implicada una reunión como si no— se pueden crear índices para acelerar la evaluación de los predicados de selección (la cláusula **where** de SQL) que sea posible que aparezcan.

Otro aspecto de las consultas que afecta a la elección de índices es la proporción relativa de operaciones de actualización y de lectura. Aunque los índices pueden acelerar las consultas, también ralentizan las actualizaciones, que se ven obligadas a realizar un trabajo adicional para mantener la exactitud de los índices.

7.10.3. Requisitos de autorización

Las restricciones de autorización también afectan al diseño de las bases de datos, ya que SQL permite que se autorice el acceso a los usuarios en función de los componentes del diseño lógico de la base de datos. Puede que haga falta descomponer un esquema de relación en dos o más esquemas para facilitar la concesión de derechos de acceso en SQL. Por ejemplo, un registro de empleados puede contener datos relativos a nóminas, funciones de los puestos y prestaciones sanitarias. Como diferentes unidades administrativas de la empresa pueden manejar cada uno de los diversos tipos de datos, algunos usuarios necesitarán acceso a los datos de las nóminas, mientras se les deniega el acceso a los datos de las funciones de los puestos de trabajo, a los de las prestaciones sanitarias, etc. Si todos esos datos se hallan en una relación, la deseada división del acceso, aunque todavía posible mediante el uso de vistas, resulta más complicada. La división de los datos, de este modo, pasa a ser todavía más crítica cuando estos se distribuyen en varios sistemas de una red informática, un aspecto que se considera en el Capítulo 19.

7.10.4. Flujos de datos y flujos de trabajo

Las aplicaciones de bases de datos suelen formar parte de una aplicación empresarial de mayor tamaño que no solo interactúa con el sistema de bases de datos, sino también con diferentes aplicaciones especializadas. Por ejemplo, en una compañía manufacturera, puede que un sistema de diseño asistido por computadora (computer-aided design, CAD) ayude al diseño de nuevos productos. Puede que el sistema CAD extraiga datos de la base de datos mediante instrucciones de SQL, procese internamente los datos, quizás interactuando con un diseñador de productos, y luego actualice la base de datos. Durante este proceso, el control de los datos puede pasar a manos de varios diseñadores de productos y de otras personas. Como ejemplo adicional, considere un informe de gastos de viaje. Lo crea un empleado que vuelve de un viaje de negocios (posiblemente mediante un paquete de software especial) y luego se envía al jefe de ese empleado, quizás a otros jefes de niveles superiores y, finalmente, al departamento de contabilidad para su pago (momento en el que interactúa con los sistemas informáticos de contabilidad de la empresa).

El término *flujo de trabajo* hace referencia a la combinación de datos y de tareas implicados en procesos como los de los ejemplos anteriores. Los flujos de trabajo interactúan con el sistema de bases de datos cuando se mueven entre los usuarios y estos llevan a cabo sus tareas en el flujo de trabajo. Además de los datos sobre los que opera el flujo de trabajo, puede que la base de datos almacene datos sobre el propio flujo de trabajo, incluidas las tareas que lo conforman y la manera en que se han de hacer llegar a los usuarios. Por tanto, los flujos de trabajo especifican una serie de consultas y de actualizaciones de la base de datos que pueden tenerse en cuenta como parte del proceso de diseño de la base de datos. En otras palabras, el modelado de la empresa no solo exige la comprensión de la semántica de los datos, sino también la de los procesos comerciales que los usan.

7.10.5. Otros elementos del diseño de bases de datos

El diseño de bases de datos no suele ser una actividad que se pueda dar por acabada en una sola vez. Las necesidades de las organizaciones evolucionan continuamente, y los datos que necesitan almacenar también evolucionan en consonancia. Durante las fases iniciales del diseño de la base de datos, o durante el desarrollo de las aplicaciones, puede que el diseñador de la base de datos se dé cuenta de que hacen falta cambios en el nivel del esquema conceptual, lógico o físico. Los cambios del esquema pueden afectar a todos los aspectos de la aplicación de bases de datos. Un buen diseño de bases de datos se anticipa a las necesidades futuras de la organización y el diseño se lleva a cabo de manera que se necesiten modificaciones mínimas a medida que evolucionen las necesidades de la organización.

Es importante distinguir entre las restricciones fundamentales que se espera sean permanentes y las que se anticipa que puedan cambiar. Por ejemplo, la restricción de que cada identificador de profesor identifique a un solo profesor es fundamental. Por otro lado, la universidad puede tener la norma de que cada profesor solo pueda estar asociado a un departamento, lo que puede cambiar más adelante si se permiten otras asociaciones. Un diseño de la base de datos que solo permita un departamento por profesor necesitaría cambios importantes si se pudiesen realizar otras asociaciones. Estas asociaciones se pueden representar añadiendo una relación extra, sin modificar la relación *profesor*, siempre que los profesores tengan solo asociación a un departamento principal; el cambio de la norma que permite más de una asociación principal puede requerir un cambio mayor en el diseño de la base de datos. Un buen diseño debería tener en cuenta no solo las normas actuales, sino que debería evitar o minimizar cambios porque se han anticipado dichos cambios, o existe una cierta probabilidad de que estos se produzcan.

Además, es probable que la empresa a la que sirve la base de datos interactúe con otras empresas y, por tanto, puede que tengan que interactuar varias bases de datos. La conversión de los datos entre esquemas diferentes es un problema importante en las aplicaciones del mundo real. Se han propuesto diferentes soluciones para este problema. El modelo de datos XML, que se estudia en el Capítulo 23, se usa mucho para representar los datos cuando estos se intercambian entre diferentes aplicaciones.

Finalmente, merece la pena destacar que el diseño de bases de datos es una actividad orientada a los seres humanos en dos sentidos: los usuarios finales son personas (aunque se sitúe alguna aplicación entre la base de datos y los usuarios finales) y el diseñador de la base de datos debe interactuar intensamente con los expertos en el dominio de la aplicación para comprender los requisitos de datos de la aplicación. Todas las personas involucradas con los datos tienen necesidades y preferencias que se deben tener en cuenta para que el diseño y la implantación de la base de datos tengan éxito en la empresa.

7.11. Resumen

- El diseño de bases de datos supone principalmente el diseño del esquema de la base de datos. El modelo de datos **entidad-relación (E-R)** es un modelo de datos muy usado para el diseño de bases de datos. Ofrece una representación gráfica adecuada para ver los datos, las relaciones y las restricciones.
- El modelo está pensado principalmente para el proceso de diseño de la base de datos. Se desarrolló para facilitar el diseño de bases de datos al permitir la especificación de un **esquema de la empresa.** Este esquema representa la estructura lógica general de la base de datos. Esta estructura general se puede expresar gráficamente mediante un **diagrama E-R.**
- Una **entidad** es un objeto que existe en el mundo real y es distinguible de otros objetos. Esa distinción se expresa asociando a cada entidad un conjunto de atributos que describen el objeto.
- Una **relación** es una asociación entre diferentes entidades. Un **conjunto de relaciones** es una colección de relaciones del mismo tipo, y un **conjunto de entidades** es una colección de entidades del mismo tipo.
- Los términos **superclave, clave candidata** y **clave primaria** se aplican a conjuntos de entidades y de relaciones al igual que a los esquemas de relación. La identificación de la clave primaria de un conjunto de relaciones requiere un cierto cuidado, ya que se compone de atributos de uno o más de los conjuntos de entidades relacionados.
- La **correspondencia de cardinalidades** expresa el número de entidades con las que otra entidad se puede asociar mediante un conjunto de relaciones.
- Un conjunto de entidades que no tiene suficientes atributos para formar una clave primaria se denomina **conjunto de entidades débiles.** Un conjunto de entidades que tiene una clave primaria se denomina **conjunto de entidades fuertes.**
- Las diferentes características del modelo E-R ofrecen al diseñador de bases de datos numerosas opciones a la hora de representar lo mejor posible la empresa que se modela. Los conceptos y los objetos pueden, en ciertos casos, representarse mediante entidades, relaciones o atributos. Ciertos aspectos de la estructura global de la empresa se pueden describir mejor usando los conjuntos de entidades débiles, la generalización, la especialización o la agregación. A menudo, el diseñador debe sopesar las ventajas de un modelo simple y compacto frente a las de otro más preciso pero más complejo.
- El diseño de una base de datos especificado en un diagrama E-R se puede representar mediante un conjunto de esquemas de relación. Para cada conjunto de entidades y para cada conjunto de relaciones de la base de datos hay un solo esquema de relación al que se le asigna el nombre del conjunto de entidades o de relaciones correspondiente. Esto forma la base para la obtención del diseño de la base de datos relacional a partir del E-R.
- La **especialización** y la **generalización** definen una relación de inclusión entre un conjunto de entidades de nivel superior y uno o más conjuntos de entidades de nivel inferior. La especialización es el resultado de tomar un subconjunto de un conjunto de entidades de nivel superior para formar un conjunto de entidades de nivel inferior. La generalización es el resultado de tomar la unión de dos o más conjuntos disjuntos de entidades (de nivel inferior) para producir un conjunto de entidades de nivel superior. Los atributos de los conjuntos de entidades de nivel superior los heredan los conjuntos de entidades de nivel inferior.
- La **agregación** es una abstracción en la que los conjuntos de relaciones (junto con sus conjuntos de entidades asociados) se tratan como conjuntos de entidades de nivel superior y pueden participar en las relaciones.
- El **lenguaje de modelado unificado (UML)** es un lenguaje de modelado muy usado. Los diagramas de clases de UML son muy utilizados para el modelado de clases, así como para otros objetivos de modelado de datos.

Términos de repaso

- Modelo de datos entidad-relación.
- Entidad y conjunto de entidades.
  - Atributos.
  - Dominio.
  - Atributos simples y compuestos.
  - Atributos de un solo valor y multivalorados.
  - Valor nulo (*null*).
  - Atributo derivado.
  - Superclave, clave candidata y clave primaria.
- Relaciones y conjuntos de relaciones.
  - Conjunto de relaciones binarias.
  - Grado del conjunto de relaciones.
  - Atributos descriptivos.
  - Superclave, clave candidata y clave primaria.
  - Rol.
  - Conjunto de relaciones recursivo.
- Diagrama E-R.
- Correspondencia de cardinalidades.
  - Relación uno a uno.
  - Relación uno a varios.
  - Relación varios a uno.
  - Relación varios a varios.
- Participación.
  - Participación total.
  - Participación parcial.
- Conjuntos de entidades débiles y fuertes.
  - Atributo discriminador.
  - Relación identificadora.
- Especialización y generalización.
  - Superclase y subclase.
  - Herencia de atributos.
  - Herencia simple y múltiple.
  - Pertenencia definida por condición y definida por el usuario.
  - Generalización disjunta y solapada.
  - Generalización total y parcial.
- Agregación.
- UML.
- Diagramas de clase en UML.

Ejercicios prácticos

7.1. Construya un diagrama E-R para una compañía de seguros de coches cuyos clientes poseen uno o más coches cada uno. Cada coche tiene asociado un valor de cero al número de accidentes registrados. Cada póliza de seguros cubre uno o más coches, y tiene uno o más pagos «premium» asociados. Cada pago es para un periodo de tiempo, y tiene asociado un plazo y la fecha en la que el pago se ha efectuado.

7.2. Considere una base de datos usada para registrar las notas que obtienen los estudiantes en diferentes exámenes de distintas asignaturas ofertadas.

a. Construya un diagrama E-R que modele los exámenes como entidades y utilice una relación ternaria para la base de datos.

b. Construya un diagrama E-R alternativo que solo utilice una relación binaria entre *estudiantes* y *asignaturas*. Hay que asegurarse de que solo existe una relación entre cada pareja formada por un estudiante y una asignatura, pero se pueden representar las notas que obtiene cada estudiante en diferentes exámenes de una asignatura.

7.3. Diseñe un diagrama E-R para almacenar los logros de su equipo deportivo favorito. Se deben almacenar los partidos jugados, el resultado de cada partido, los jugadores de cada partido y las estadísticas de cada jugador en cada partido. Las estadísticas resumidas se deben representar como atributos derivados.

7.4. Suponga un diagrama E-R en el que el mismo conjunto de entidades aparezca varias veces. ¿Por qué permitir esa redundancia es una mala práctica que se debe evitar siempre que sea posible?

7.5. Los diagramas E-R se pueden ver como grafos. ¿Qué significa lo siguiente en términos de la estructura del esquema de una empresa?

a. El grafo es inconexo.

b. El grafo tiene un ciclo.

7.6. Considere la representación de una relación ternaria mediante relaciones binarias como se describió en la Sección 7.7.3 que se muestra en la Figura 7.27b (no se muestran los atributos).

a. Muestre un ejemplar simple de E, A, B, C, R_A, R_B y R_C que no pueda corresponder a ningún ejemplar de A, B, C y R.

b. Modifíquese el diagrama E-R de la Figura 7.27b para introducir restricciones que garanticen que cualquier ejemplar E, A, B, C, R_A, R_B y R_C que satisfaga las restricciones corresponda a algún ejemplar de A, B, C y R.

c. Modifique la traducción anterior para manejar las restricciones de participación total sobre las relaciones ternarias.

d. La representación anterior exige que se cree un atributo de clave primaria para E. Muéstrese la manera de tratar E como un conjunto de entidades débiles de forma que no haga falta ningún atributo de clave primaria.

7.7. Los conjuntos de entidades débiles siempre se pueden convertir en conjuntos de entidades fuertes simplemente añadiendo a sus atributos los atributos de clave primaria del conjunto de entidades identificadoras. Describa el tipo de redundancia que se produce al hacerlo.

7.8. Considere una relación como *secc_asignatura*, generada desde una relación varios a uno *secc_asignatura*. ¿Las restricciones de clave primaria y externa creadas en la relación duerzan a la restricción de cardinalidad varios a uno? Explique por qué.

7.9. Suponga que la relación *tutor* es de uno a uno. ¿Qué restricciones extra son necesarias en la relación *tutor* para asegurar que se cumple la restricción de cardinalidad uno a uno?

7.10. Considere una relación varios a uno R entre los conjuntos de entidades A y B. Suponga que la relación creada de R se combina con una relación creada de A. En SQL, los atributos participantes en la restricción de clave externa puede ser *null*. Explique cómo se puede cumplir la restricción de participación total de A en R utilizando restricciones **not null** en SQL.

7.11. En SQL, las restricciones de clave externa solo pueden referenciar los atributos de clave primaria de la relación referenciada, u otros atributos declarados como superclave usando la restricción **unique.** En consecuecia, las restricciones de participación total en una relación de varios a varios (o en el lado «uno» de una relación de uno a varios) no se puede forzar a cumplir en las relaciones creadas a partir de una relación, mediante el uso de restricciones de clave primaria, de clave externa o *not null*.

a. Explique por qué.

b. Explique cómo forzar las restricciones de participación total usando restricciones check complejas o aserciones (véase la Sección 4.4.7). (Desafortunadamente estas características no están disponibles en ninguna de las bases de datos más utilizadas actualmente).

7.12. La Figura 7.28 muestra una estructura reticular de generalización y especialización (no se muestran los atributos). Para los conjuntos de entidades A, B y C explique cómo se heredan los atributos desde los conjuntos de entidades de nivel superior X e Y. Explique la manera de manejar el caso de que un atributo de X tenga el mismo nombre que algún atributo de Y.

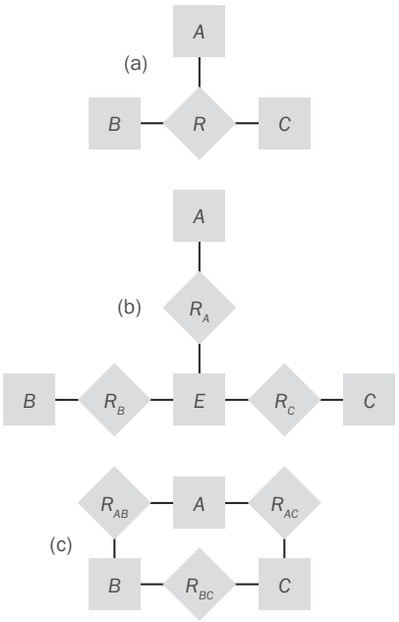


Figura 7.27. Diagrama E-R para los Ejercicios prácticos 7.6 y 7.24.

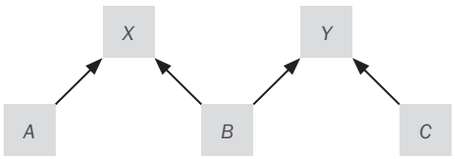


Figura 7.28. Diagrama E-R para el Ejercicio práctico 7.12.

7.13. Cambios temporales: un diagrama E-R normalmente modela el estado de una empresa en un momento del tiempo. Suponga que se desea hacer un seguimiento de *cambios temporales*, es decir, los cambios de los datos con el tiempo. Por ejemplo, Zhang puede que haya sido estudiante entre el 1 de septiembre de 2005 y el 31 de mayo de 2009, mientras que Shankar puede haber tenido como tutor al profesor Einstein desde el 31 de mayo de 2008 al 5 de diciembre de 2008 y de nuevo desde el 1 de junio de 2009 al 5 de enero de 2010. De forma similar, los valores de un atributo de una entidad o de una relación, como el *nombre* de la asignatura o los *créditos* de la misma, el *sueldo*, el *nombre* de un *profesor* y el *tot_créd* de un *estudiante,* pueden cambiar con el tiempo.

Una forma de modelar los cambios temporales es la siguiente: se define un nuevo tipo de dato que se denomine **tiempo válido,** que es un intervalo de tiempo o un conjunto de intervalos de tiempo. Entonces, se asocia un atributo *tiempo_válido* a cada una de las entidades y relaciones, registrando los periodos de tiempo durante los cuales la entidad o la relación es válida. La fecha final de un intervalo puede ser infinita; por ejemplo, si Shankar se convierte en estudiante el 2 de septiembre de 2008, y aún sigue siendo estudiante, la ficha final se puede representar como infinito para la entidad Shankar. De forma similar, se modelan los atributos que van a cambiar con el tiempo como un conjunto de valores, cada uno con su propio *tiempo_válido*.

a. Dibuje un diagrama E-R con las entidades *estudiante* y *profesor*, y la relación *tutor*, con las extensiones anteriores para hacer un seguimiento de los cambios temporales.

b. Convierta el diagrama E-R anterior en un conjunto de relaciones.

Debería quedar claro que estos conjuntos de relaciones que se han generado resultan bastante complejos, lo que conduce a dificultades en las tareas al escribir las consultas en SQL. Otra forma de hacerlo que se utiliza más es ignorar los cambios temporales cuando se diseña en modelo E-R (en particular los cambios en el tiempo de los valores de los atributos), y modificar las relaciones generadas a partir del modelo E-R para hacer el seguimiento de los cambios temporales, como se verá más adelante en la Sección 8.9.

Ejercicios

7.14. Explique las diferencias entre los términos clave primaria, clave candidata y superclave.

7.15. Construya un diagrama E-R para un hospital con un conjunto de pacientes y un conjunto de médicos. Asóciese con cada paciente un registro de las diferentes pruebas y exámenes realizados.

7.16. Construya los esquemas de relación adecuados para cada uno de los diagramas E-R de los Ejercicios prácticos 7.1 a 7.3.

7.17. Extienda el diagrama E-R del Ejercicio práctico 7.3 para que almacene la misma información para todos los equipos de una liga.

7.18. Explique las diferencias entre los conjuntos de entidades débiles y los conjuntos de entidades fuertes.

7.19. Se puede convertir cualquier conjunto de entidades débiles en un conjunto de entidades fuertes simplemente añadiendo los atributos apropiados. ¿Por qué, entonces, se tienen conjuntos de entidades débiles?

7.20. Considere el diagrama E-R de la Figura 7.29, que modela una librería en línea.

a. Liste los conjuntos de entidades y sus claves primarias.

b. Supóngase que la librería añade discos Blu-Ray y vídeos descargables a su colección. El mismo elemento puede estar presente en uno o en ambos formatos, con diferentes precios. Extienda el diagrama E-R para modelar esta adición, ignorando el efecto sobre las cestas de la compra.

c. Extienda ahora el diagrama E-R mediante la generalización para modelar el caso en que una cesta de la compra pueda contener cualquier combinación de libros, discos Blu-Ray o vídeos descargables.

7.21. Diseñe una base de datos para una compañía de automóviles para proporcionar a los concesionarios asistencia en el mantenimiento de la información de los registros de clientes y el inventario del concesionario, y ayudar a los vendedores a realizar las peticiones de coches.

Los vehículos están identificados por un número de identificación de vehículo (NIV). Cada vehículo pertenece a un modelo concreto de una determinada marca de coche que vende la compañía (por ejemplo, el XF es un modelo de coche de la marca Jaguar de Tata Motors). Cada modelo se puede ofrecer con distintas opciones, pero un coche dado solo puede tener algunas (o ninguna) de las opciones disponibles. La base de datos debe guardar información de los modelos, marcas y opciones, así como sobre cada uno de los concesionarios, clientes y coches.

El diseño debería incluir un diagrama E-R, un conjunto de esquemas relacionales y una lista de restricciones, incluyendo restricciones de clave primaria y externa.

7.22. Diseñe una base de datos para una compañía de logística internacional de paquetería (como DHL o FedEx). La base de datos debe poder realizar un seguimiento de los clientes (quién envía los paquetes) y los clientes (quien recibe los paquetes); algunos clientes pueden estar en las dos partes.

Se debe poder identificar y trazar cada uno de los paquetes, por lo que la base de datos debe poder guardar la ubicación de un paquete y su histórico de ubicaciones. Las ubicaciones pueden incluir camiones, aviones, aeropuertos y almacenes.

El diseño debería incluir un diagrama E-R, un conjunto de esquemas relacionales y una lista de restricciones, incluyendo restricciones de clave primaria y externa.

7.23. Diseñe la base de datos de una línea aérea. La base de datos debe guardar información de los clientes y sus reservas, los vuelos y su estado, la asignación de asientos en cada uno de los vuelos, así como el plan y ruta de los futuros vuelos.

El diseño debería incluir un diagrama E-R, un conjunto de esquemas relacionales y una lista de restricciones, incluyendo restricciones de clave primaria y externa.

7.24. En la Sección 7.7.3 se representó una relación ternaria (que se repite en la Figura 7.27a) mediante relaciones binarias, como se muestra en la Figura 7.27b. Considere la alternativa mostrada en la Figura 7.27c. Explique las ventajas relativas de estas dos representaciones alternativas de una relación ternaria mediante relaciones binarias.

7.25. Considere los esquemas de relación mostrados en la Sección 7.6, que se generaron a partir del diagrama E-R de la Figura 7.15. Para cada esquema, especifique las restricciones de clave externa que hay que crear, si es que hay que crear alguna.

7.26. Diseñe una jerarquía de especialización–generalización para una compañía de venta de vehículos de motor. La compañía vende motocicletas, coches, furgonetas y autobuses. Justifique la ubicación de los atributos en cada nivel de la jerarquía.

Explique el motivo por el que no se deben ubicar en un nivel superior o inferior.

7.27. Explique la diferencia entre las restricciones definidas por condición y las definidas por el usuario. ¿Cuáles de estas restricciones pueden comprobar el sistema automáticamente?

7.28. Explique la diferencia entre las restricciones disjuntas y las solapadas.

7.29. Explique la diferencia entre las restricciones totales y las parciales.

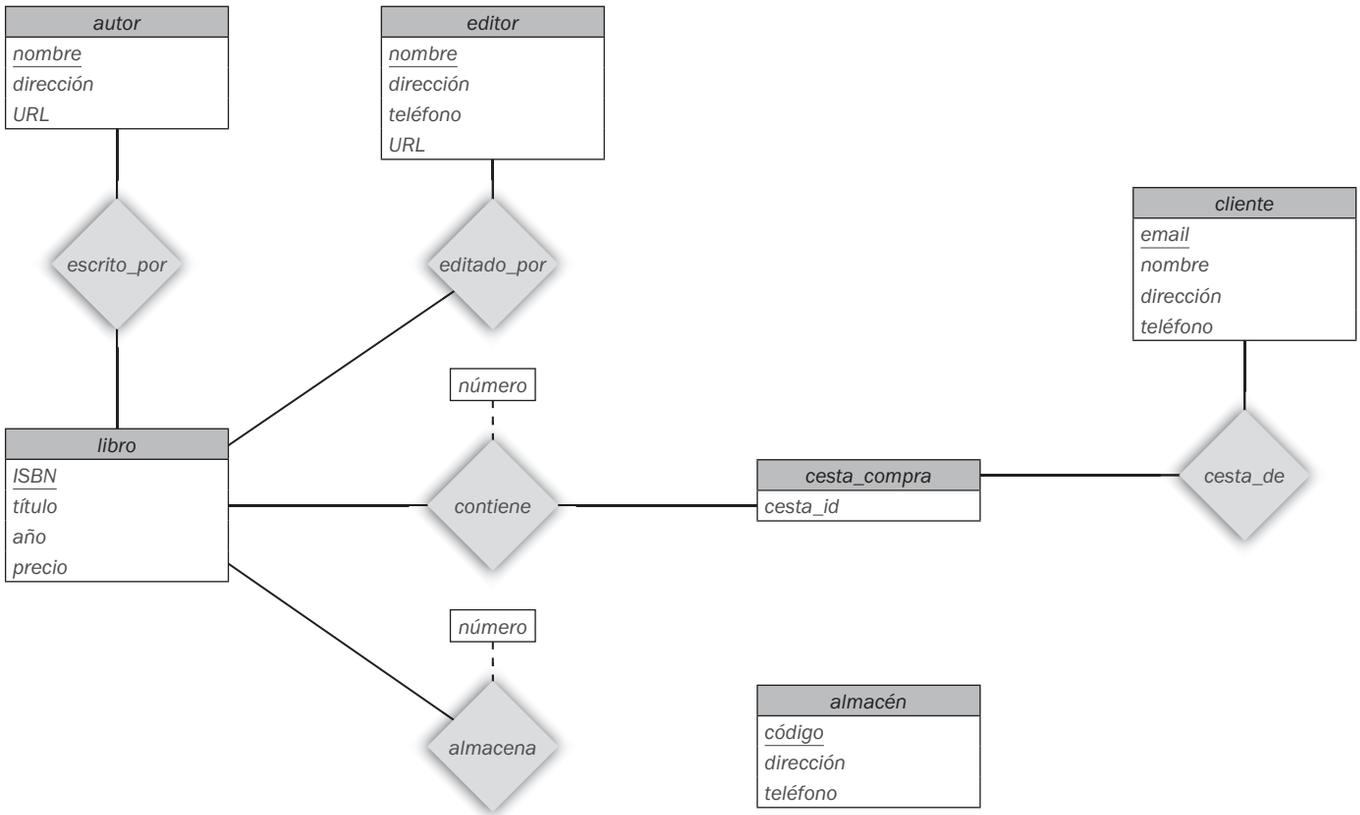


Figura 7.29. Diagrama E-R para el Ejercicio 7.20.

Herramientas

Muchos sistemas de bases de datos ofrecen herramientas para el diseño de bases de datos que soportan los diagramas E-R. Estas herramientas ayudan al diseñador a crear diagramas E-R y pueden crear automáticamente las tablas correspondientes de la base de datos. Véanse las notas bibliográficas del Capítulo 1 para consultar referencias de sitios web de fabricantes de bases de datos.

También hay distintas herramientas de modelado de datos independientes de las bases de datos que soportan los diagramas E-R y los diagramas de clases de UML. La herramienta de dibujo Dia, que está disponible como freeware, dispone de diagramas E-R y de diagramas de clase de UML. Entre las herramientas comerciales están Rational Rose (www.ibm.com/software/rational), Microsoft Visio (véase www.microsoft.com/office/visio), CA's ERwin (www.ca.com/us/data-modeling.aspx), Poseidon for UML (www.gentleware.com), y SmartDraw (www.smartdraw.com).

Notas bibliográficas

El modelo de datos E-R fue introducido por Chen [1976]. Teorey et ál. [1986] presentaron una metodología de diseño lógico para las bases de datos relacionales que usa el modelo E-R extendido. La norma de definición de la integración para el modelado de información (IDEF1X, Integration Definition for Information Modeling) IDEF1X [1993], publicado por el Instituto Nacional de Estados Unidos para Normas y Tecnología (United States National Institute of Standards and Technology, NIST) definió las normas para los diagramas E-R. No obstante, hoy en día se usa una gran variedad de notaciones E-R.

Thalheim [2000] proporciona un tratamiento detallado, pero propio de un libro de texto, de la investigación en el modelado E-R. En los libros de texto, Batini et ál. [1992] y Elmasri y Navathe [2003] ofrecen explicaciones básicas. Davis et ál. [1983] proporcionan una colección de artículos sobre el modelo E-R.

En 2009 la versión más reciente de UML era la 2.2, con la versión 2.3 cerca de la adopción final. Véase www.uml.org para obtener más información sobre las normas y las herramientas de UML.

Diseño de bases de datos y el modelo E-R

08

En este capítulo se considera el problema de diseñar el esquema de una base de datos relacional. Muchos de los problemas que conlleva son parecidos a los de diseño que se han considerado en el Capítulo 7 en relación con el modelo E-R.

En general, el objetivo del diseño de una base de datos relacional es la generación de un conjunto de esquemas de relación que permita almacenar la información sin redundancias innecesarias, pero que también permita recuperarla fácilmente. Esto se consigue mediante el diseño de esquemas que se hallen en la forma *normal adecuada*. Para determinar si el esquema de una relación se halla en una de las formas normales deseables es necesario obtener información sobre la empresa real que se está modelando con la base de datos. Parte de esa información se halla en un diagrama E-R bien diseñado, pero puede ser necesaria información adicional sobre la empresa.

En este capítulo se introduce un enfoque foral al diseño de bases de datos relacionales basado en el concepto de dependencia funcional. Posteriormente se definen las formas normales en términos de las dependencias funcionales y de otros tipos de dependencias de datos. En primer lugar, sin embargo, se examina el problema del diseño relacional desde el punto de vista de los esquemas derivados de un diseño entidad-relación dado.

8.1. Características de los buenos diseños relacionales

El estudio del diseño entidad-relación llevado a cabo en el Capítulo 7 ofrece un excelente punto de partida para el diseño de bases de datos relacionales. Ya se vio en la Sección 7.6 que es posible generar directamente un conjunto de esquemas de relación a partir del diseño E-R. Evidentemente, la adecuación (o no) del conjunto de esquemas resultante depende, en primer lugar, de la calidad del diseño E-R. Más adelante en este capítulo se estudiarán maneras precisas de evaluar la adecuación de los conjuntos de esquemas de relación. No obstante, es posible llegar a un buen diseño empleando conceptos que ya se han **estudiado.**

Para facilitar las referencias, en la Figura 8.1 se repiten los esquemas de la base de datos de la universidad.

aula (*edificio*, *número_aula*, *capacidad*)
departamento (*nombre_dept*, *edificio*, *presupuesto*)
asignatura (*asignatura_id*, *nombre_asig*, *nombre_dept*, *créditos*)
profesor (*ID*, *nombre*, *nombre_dept*, *sueldo*)
sección (*asignatura_id*, *secc_id*, *semestre*, *año*, *edificio*, *número_aula*, *franja_horaria_id*)
enseña (*ID*, *asignatura_id*, *secc_id*, *semestre*, *año*)
estudiante (*ID*, *nombre*, *nombre_dept*, *tot_cred*)
matricula (*ID*, *asignatura_id*, *secc_id*, *semestre*, *año*, *nota*)
tutor (*e_ID*, *p_ID*)
franja_horaria (*franja_horaria_id*, *día*, *hora_inicio*, *hora_fin*)
prerreq (*asignatura_id*, *prerreq_id*)

Figura 8.1. Esquema de base de datos de la universidad.

8.1.1. Alternativa de diseño: esquemas grandes

A continuación se examinan las características de este diseño de base de datos relacional y algunas alternativas. Supóngase que, en lugar de los esquemas *profesor* y *departamento*, se considerase el esquema:

profesor_dept (ID, nombre, sueldo, nombre_dept, edificio, presupuesto)

Esto representa el resultado de la reunión natural de las relaciones correspondientes a *profesor* y *departamento*. Parece una buena idea, ya que es posible expresar algunas consultas empleando menos reuniones, hasta que se considera detenidamente la realidad de la universidad que condujo a nuestro diseño E-R.

Considérese el ejemplar de la relación *profesor_dept* que se ve en la Figura 8.2. Téngase en cuenta que hay que repetir la información de departamento («edificio» y «presupuesto») para cada profesor del departamento. Por ejemplo, la información sobre el departamento Informática (Taylor, 100000) se incluye en las tuplas de los profesores Katz, Srinivasan y Brandt.

ID	*nombre*	*sueldo*	*nombre_dept*	*edificio*	*presupuesto*
22222	Einstein	95000	Física	Watson	70000
12121	Wu	90000	Finanzas	Painter	120000
32343	El Said	60000	Historia	Painter	50000
45565	Katz	75000	Informática	Taylor	100000
98345	Kim	80000	Electrónica	Taylor	85000
76766	Crick	72000	Biología	Watson	90000
10101	Srinivasan	65000	Informática	Taylor	100000
58583	Califieri	62000	Historia	Painter	50000
83821	Brandt	92000	Informática	Taylor	100000
15151	Mozart	40000	Música	Packard	80000
33456	Gold	87000	Física	Watson	70000
76543	Singh	80000	Finanzas	Painter	120000

Figura 8.2. Tabla *profesor_dept*.

Es importante que todas estas tuplas coincidan en la cuantía del presupuesto ya que si no la base de datos se volvería inconsistente. En el diseño original que utilizaba *profesor* y *departamento*, se almacenaba la cuantía de cada presupuesto una única vez. Ello sugiere que el uso de *profesor_dept* es una mala idea, ya que el presupuesto se almacena de forma redundante y se corre el riesgo de que algún usuario actualice el presupuesto en una de las tuplas pero no en todas y por tanto se cree una inconsistencia.

Aunque se decidiese mantener el problema de la redundancia, existe otro problema con el esquema *profesor_dept*. Supóngase que se crea un nuevo departamento en la universidad. En el diseño alternativo anterior, no se puede representar directamente la información de un departamento *(nombre_dept, edificio, presupuesto)* a no ser que ese departamento tenga al menos un profesor de la universidad. Es así

porque las tuplas de la tabla *profesor_dept* requieren valores para *ID*, *nombre* y *sueldo*. Esto significa que no se puede registrar información sobre el nuevo departamento creado hasta que se contrate al primer profesor para dicho departamento. En el esquema anterior, el esquema *departamento* podía tratar esta situación, pero en el diseño revisado se debería crear una tupla con un valor *null* (nulo) en *edificio* y en *presupuesto*. En algunos casos los valores *null* generan problemas, como se verá al tratar SQL. Sin embargo, si se decide que no supone un problema en este caso, vamos a intentar utilizar el nuevo esquema.

8.1.2. Alternativa de diseño: esquemas pequeños

Supóngase nuevamente que, de algún modo, se ha comenzado a trabajar con el esquema *profesor_dept*. ¿Cómo reconocer la necesidad de repetición de la información y de dividirla en dos esquemas, *profesor* y *departamento*?

Al observar el contenido de la relación en el esquema *profesor_dept*, se puede observar la repetición de la información que surge de realizar el listado del edificio y el presupuesto una vez para cada profesor de un departamento. Sin embargo, se trata de un proceso en el que no se puede confiar. Las bases de datos reales tienen gran número de esquemas y un número todavía mayor de atributos. El número de tuplas puede ser del orden de millones o superior. Descubrir la repetición puede resultar costoso. Hay un problema más fundamental todavía en este enfoque. No permite determinar si la carencia de repeticiones es meramente un caso especial «afortunado» o la manifestación de una regla general. En nuestro ejemplo, ¿cómo podríamos *saber* que en nuestra universidad, todos los departamentos (identificados por su nombre de departamento) *tienen que* encontrarse en un único edificio y deben tener un único presupuesto? ¿El hecho de que el presupuesto del departamento Informática aparezca tres veces con el mismo presupuesto es solo una coincidencia? Estas preguntas no se pueden responder sin volver a la propia universidad y comprender sus reglas de funcionamiento. En concreto, hay que averiguar si la universidad exige que todos los departamentos (identificados por su nombre de departamento) *deben* tener un único edificio y un único valor de presupuesto.

En el caso de *profesor_dept*, el proceso de creación del diseño E-R logró evitar la creación de ese esquema. Sin embargo, esta situación fortuita no se produce siempre.

Por tanto, hay que permitir que el diseñador de la base de datos especifique normas como «cada valor de *nombre_dept* corresponde, como máximo, a un *presupuesto*», incluso en los casos en los que *nombre_dept* no sea la clave primaria del esquema en cuestión. En otras palabras, hay que escribir una norma que diga «si hay un esquema (*nombre_dept*, *presupuesto*), entonces *nombre_dept* puede hacer de clave primaria». Esta regla se especifica como la **dependencia funcional.**

$$nombre_dept \rightarrow presupuesto$$

Dada esa regla, ya se tiene suficiente información para reconocer el problema del esquema *profesor_dept*. Como *nombre_dept* no puede ser la clave primaria de *profesor_dept* (porque puede que cada departamento necesite varias tuplas de la relación del esquema *profesor_dept*), tal vez haya que repetir el importe del presupuesto.

Observaciones como estas y las reglas (especialmente las dependencias funcionales) que generan permiten al diseñador de la base de datos reconocer las situaciones en que hay que dividir, o *descomponer*, un esquema en dos o más. No es difícil comprender que la manera correcta de descomponer *profesor_dept* es dividirlo en *profesor* y *departamento*, como en el diseño original. Hallar la descomposición correcta es mucho más difícil para esquemas con gran número de atributos y varias dependencias funcionales. Para trabajar con ellos hay que confiar en una metodología formal que se desarrolla más avanzado este capítulo.

No todas las descomposiciones de los esquemas resultan útiles. Considérese el caso extremo en que todo lo que tengamos sean esquemas con un solo atributo. No se puede expresar ninguna relación interesante de ningún tipo. Considérese ahora un caso menos extremo en el que se decida descomponer el esquema *empleado* (véase Sección 7.8):

empleado (*ID, nombre, calle, ciudad, sueldo*)

en los siguientes dos esquemas:

empleado1 (*ID, nombre*)
empleado2 (*nombre, calle, ciudad, sueldo*)

El problema con esta descomposición surge de la posibilidad de que la empresa tenga dos empleados que se llamen igual. Esto no es improbable en la práctica, ya que muchas culturas tienen algunos nombres muy populares Por supuesto, cada persona tiene un identificador de empleado único, que es el motivo de que se pueda utilizar *ID* como clave primaria. A modo de ejemplo, supóngase que dos empleados llamados Kim, trabajan para la misma universidad y tienen las tuplas siguientes de la relación del esquema *empleado* del diseño original:

(57766, Kim, Main, Perryridge, 75000)
(98776, Kim, North, Hampton, 67000)

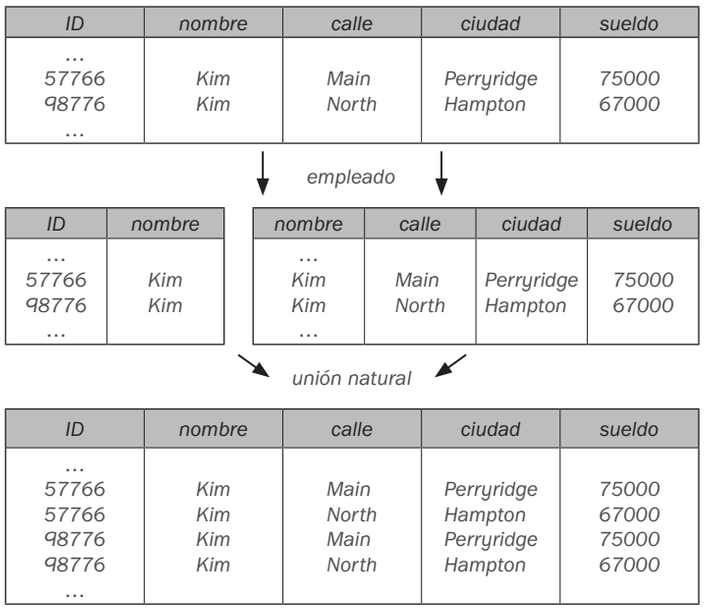


Figura 8.3. Pérdida de información debida a una mala descomposición.

La Figura 8.3 muestra estas tuplas, las tuplas resultantes al utilizar los esquemas procedentes de la descomposición y el resultado que se obtendría si se intentara volver a generar las tuplas originales mediante una reunión natural. Como se ve en la figura, las dos tuplas originales aparecen en el resultado junto con dos tuplas nuevas que mezclan de manera incorrecta valores de fechas correspondientes a los dos empleados llamados Kim. Aunque se dispone de más tuplas, en realidad se tiene menos información en el sentido siguiente: se puede indicar que un determinado número de teléfono y una fecha de contratación dada corresponden a alguien llamado Kim, pero no se puede distinguir a cuál de ellos. Por tanto, la descomposición propuesta es incapaz de representar algunos datos importantes de los empleados de la universidad. Evidentemente, es preferible evitar este tipo de descomposiciones. Estas descomposiciones se denominan **descomposiciones con pérdidas,** y, a la inversa, aquellas que no tienen pérdidas se denominan **descomposiciones sin pérdidas.**

8.2. Dominios atómicos y la primera forma normal

El modelo E-R permite que los conjuntos de entidades y los de relaciones tengan atributos con algún tipo de subestructura. Concretamente, permite atributos como *número_teléfono* de la Figura 7.11 y atributos compuestos (como el atributo *dirección* con los atributos *calle*, *ciudad*, *estado*, y *CP*). Cuando se crean tablas a partir de los diseños E-R que contienen ese tipo de atributos, se elimina esa subestructura. Para los atributos compuestos, se deja que cada atributo componente sea un atributo de pleno derecho. Para los atributos multivalorados, se crea una tupla por cada elemento del conjunto multivalorado.

En el modelo relacional se formaliza esta idea de que los atributos no tienen subestructuras. Un dominio es **atómico** si se considera que los elementos de ese dominio son unidades indivisibles. Se dice que el esquema de la relación R está en la **primera forma normal** (1FN) si los dominios de todos los atributos de R son atómicos.

Los conjuntos de nombres son ejemplos de valores no atómicos. Por ejemplo, si el esquema de la relación *empleado* incluyera el atributo *hijos* cuyos elementos de dominio fueran conjuntos de nombres, el esquema no estaría en la primera forma normal.

Los atributos compuestos, como el atributo *dirección* con los atributos componentes *calle*, *ciudad*, *estado*, y *CP* tienen también dominios no atómicos.

Se da por supuesto que los números enteros son atómicos, por lo que el conjunto de los números enteros es un dominio atómico; el conjunto de todos los conjuntos de números enteros es un dominio no atómico. La diferencia estriba en que normalmente no se considera que los enteros tengan subpartes, pero sí se considera que las tienen los conjuntos de enteros —es decir, los enteros que componen el conjunto—. Pero lo importante no es lo que sea el propio dominio, sino el modo en que se utilizan los elementos de ese dominio en la base de datos. El dominio de todos los enteros sería no atómico si se considerara que cada entero es una lista ordenada de cifras.

Como ilustración práctica del punto anterior, considérese una organización que asigna a los empleados números de identificación de la manera siguiente: las dos primeras letras especifican el departamento y las cuatro cifras restantes son un número único para cada empleado dentro de ese departamento. Ejemplos de estos números pueden ser «CS001» y «EE1127». Estos números de identificación pueden dividirse en unidades menores y, por tanto, no son atómicos. Si el esquema de la relación tuviera un atributo cuyo dominio consistiera en números de identificación así codificados, el esquema no se hallaría en la primera forma normal.

Cuando se utilizan números de identificación de este tipo, se puede averiguar el departamento de cada empleado escribiendo código que analice la estructura de los números de identificación. Ello exige una programación adicional, y la información queda codificada en el programa de aplicación en vez de en la base de datos. Surgen nuevos problemas si se utilizan esos números de identificación como claves primarias: cada vez que un empleado cambie de departamento, habrá que modificar su número de identificación en todos los lugares en que aparezca, lo que puede constituir una tarea difícil; en caso contrario, el código que interpreta ese número dará un resultado erróneo.

De lo dicho hasta este momento, puede parecer que nuestro uso de los identificadores de asignatura como «CS-101», donde «CS» indica departamento de Informática, significa que el dominio de identificadores de asignatura no es atómico. Este dominio no es atómico en cuanto a las personas que utilizan este sistema se refiere. Sin embargo, la aplicación de la base de datos trata al dominio como atómico, siempre que no intente dividir el identificador e interpretar las partes del mismo como una abreviatura del departamento. El esquema de la asignatura guarda el nombre del departamento como un atributo distinto, y la aplicación de la base de datos puede utilizar el valor de este atributo para encontrar el departamento de la asignatura, en lugar de interpretar los caracteres del propio identificador de asignatura. Por tanto, se puede considerar que el esquema de nuestra universidad se encuentra en primera forma normal.

El empleo de atributos de conjunto puede dar lugar a diseños con acúmulo redundante de datos, lo que a su vez genera incosistencias. Por ejemplo, en lugar de representar la relación entre los profesores y las secciones como la relación independiente *enseña*, puede que el diseñador de bases de datos esté tentado a almacenar un conjunto de identificadores de sección de asignaturas con cada profesor y un conjunto de identificadores de profesor con cada sección. (Las claves primarias de *sección* y *profesor* se utilizan como identificadores). Siempre que se modifiquen los datos del profesor que enseña en una sección, habrá que llevar a cabo la actualización en dos lugares: en el conjunto de profesores de la sección y en el conjunto de secciones del profesor. No llevar a cabo esas dos actualizaciones puede dejar la base de datos en un estado inconsistente. Guardar solo uno de esos conjuntos evitaría información repetida, pero complicaría algunas consultas, y además, no está claro a cuál de los dos habría que mantener.

Algunos tipos de valores no atómicos pueden resultar útiles, aunque deben utilizarse con cuidado. Por ejemplo, los atributos con valores compuestos suelen resultar útiles, y los atributos con el valor determinado por un conjunto también resultan útiles en muchos casos, que es el motivo por el que el modelo E-R los soporta. En muchos dominios en los que las entidades tienen una estructura compleja, la imposición de la representación en la primera forma normal supone una carga innecesaria para el programador de las aplicaciones, que tiene que escribir código para convertir los datos a su forma atómica. También hay sobrecarga en tiempo de ejecución por la conversión de los datos de su forma habitual a su forma atómica. Por tanto, el soporte de los valores no atómicos puede resultar muy útil en ese tipo de dominios. De hecho, los sistemas modernos de bases de datos soportan muchos tipos de valores no atómicos, como se verá en el Capítulo 22. Sin embargo, en este capítulo nos limitamos a las relaciones en la primera forma normal y, por tanto, todos los dominios son atómicos.

8.3. Descomposición mediante dependencias funcionales

En la Sección 8.1 se indicó que hay una metodología formal para evaluar si un esquema relacional debe descomponerse. Esta metodología se basa en los conceptos de clave y de dependencia funcional.

Para ver los algoritmos de diseño de bases de datos relacionales, necesitamos hablar de relaciones arbitrarias y sus esquemas, en lugar de tratar solamente sobre un ejemplo. Recordando la introducción al modelo relacional del Capítulo 2, vamos a resumir la notación empleada.

- En general, se utilizan letras griegas para los conjuntos de atributos (por ejemplo, α). Se utilizan letras latinas minúsculas seguidas de una mayúscula entre paréntesis para referirse a un esquema de relación [por ejemplo, r (R)]. Se usa la notación r (R) para *mostrar que el esquema es para la relación r,* con R indicando el conjunto de atributos; pero algunas veces simplifica la notación para utilizar solo R cuando el nombre de la relación no es importante.

  Por supuesto que un esquema de relación es un conjunto de atributos, pero no todos los conjuntos de atributos son esquemas. Cuando se utiliza una letra griega minúscula se refiere a un conjunto de atributos que pueden ser o no un esquema. Se utiliza una letra latina cuando se quiere indicar que el conjunto de atributos es definitivamente un esquema.

- Cuando un conjunto de atributos es una superclave, se indica con K. Una superclave pertenece a un esquema de relación específico, por lo que se utiliza la terminología «K es una superclave de $r(R)$».
- Se utiliza un nombre en minúsculas para las relaciones. En nuestro ejemplo, se intenta que los nombres sean realistas (por ejemplo, *profesor*), mientras que en nuestras definiciones y algoritmos se sugiere utilizar solo letras como r.
- Una relación, por supuesto, tiene un valor concreto en un momento dado, al que nos referimos como ejemplar y se usa el término «ejemplar de r». Cuando queda claro que se refiere a un ejemplar, se utiliza simplemente el nombre de la relación (por ejemplo, r).

8.3.1. Claves y dependencias funcionales

Una base de datos modela un conjunto de entidades y relaciones del mundo real. Normalmente existen diversas restricciones (reglas) que se aplican sobre los datos del mundo real. Por ejemplo, algunas de las restricciones que se espera que existan en la base de datos de una universidad son:

1. Los estudiantes y los profesores se identifican de forma única por su ID.
2. Los estudiantes y los profesores tienen un único nombre.
3. Los profesores y los estudiantes están (en general) asociados a un único departamento.[1]
4. Todos los departamentos tienen un único valor de presupuesto, y solo un edificio asociado.

Un ejemplar de relación que satisfaga todas las relaciones del mundo real de este tipo se llama ejemplar legal de relación; un **ejemplar legal** de una base de datos es aquel en el que todos los ejemplares de relación son ejemplares legales.

Algunas de las restricciones más habituales del mundo real se pueden representar formalmente como claves (superclaves, claves candidatas y claves primarias), o como dependencias funcionales, que se definen a continuación.

En la Sección 2.3, se definió la noción de *superclave* como conjunto de uno o más atributos que, tomados colectivamente, permiten identificar unívocamente una tupla de la relación. Reescribimos esta definición de la siguiente forma: sea $r(R)$ un esquema de relación. Un subconjunto K de R es una **superclave** de $r(R)$ si en cualquier ejemplar legal de $r(R)$, para todos los pares t_1 y t_2 de tuplas en el ejemplar de r, si $t_1 \neq t_2$, entonces $t_1[K] \neq t_2[K]$. Es decir, no puede haber dos tuplas en una instancia legal de una relación $r(R)$ que tengan los mismos valores en el conjunto de atributos K. Claramente, si no existen dos tuplas en r que tengan los mismos valores en K, entonces un valor K identifica unívocamente una tupla en r.

Siempre que una superclave sea un conjunto de atributos que identifique unívocamente una tupla, una dependencia funcional nos permite expresar restricciones que identifiquen unívocamente los valores de ciertos atributos. Supóngase un esquema de relación $r(R)$, y sea $\alpha \subseteq R$ y $\beta \subseteq R$.

- Dado un ejemplar de $r(R)$, se dice que el ejemplar **satisface la dependencia funcional** $\alpha \rightarrow \beta$ si para todos los pares de tuplas t_1 y t_2 en los ejemplares tales que $t_1[\alpha] = t_2[\alpha]$, también se cumple que $t_1[\beta] = t_2[\beta]$.
- Se dice que una dependencia funcional $\alpha \rightarrow \beta$ **cumple** con el esquema $r(R)$ si para cada ejemplar legal de $r(R)$ satisface la dependencia funcional.

Si utilizamos la notación de dependencia funcional, se dice que K es una *superclave de* $r(R)$ si la dependencia funcional $K \rightarrow R$ cumple con $r(R)$. En otras palabras, K es una superclave si para todos los ejemplares de $r(R)$ y para cada par de tuplas t_1 y t_2 del ejemplar, siempre que $t_1[K] = t_2[K]$, también se cumple que $t_1[R] = t_2[R]$ (es decir, $t_1 = t_2$).[2]

Las dependencias funcionales permiten expresar las restricciones que no se pueden expresar con superclaves. En la Sección 8.1.2 se consideró el esquema:

profesor_dept (ID, nombre, sueldo, nombre_dept, edificio, presupuesto)

en el que se cumple la dependencia funcional *nombre_dept* → *presupuesto*, ya que por cada departamento (identificado por *nombre_dept*) existe un único presupuesto.

El hecho de que el par de atributos (*ID, nombre_dept*) forme una superclave para *profesor_dept* se denota escribiendo:

ID, nombre_dept → nombre, sueldo, edificio, presupuesto

Las dependencias funcionales se emplean de dos maneras:

1. Para probar las relaciones y ver si satisfacen un conjunto dado F de dependencias funcionales.
2. Para especificar las restricciones del conjunto de relaciones legales. Así, *solo* habrá que preocuparse de aquellas relaciones que satisfagan un conjunto dado de dependencias funcionales. Si se desea restringirse a las relaciones del esquema $r(R)$ que satisfacen un conjunto F de dependencias funcionales, se dice que F se **cumple** en $r(R)$.

Considérese la relación r de la Figura 8.4, para ver las dependencias funcionales que se satisfacen. Obsérvese que se satisface $A \rightarrow C$. Hay dos tuplas que tienen un valor para A de a_1. Estas tuplas tienen el mismo valor de C — por ejemplo, c_1. De manera parecida, las dos tuplas con un valor de a_2 para A tienen el mismo valor de C, c_2. No hay más pares de tuplas diferentes que tengan el mismo valor de A. Sin embargo, la dependencia funcional $C \rightarrow A$ no se satisface. Para ver esto, considérense las tuplas $t_1 = (a_2, b_3, c_2, d_3)$ y $t_2 = (a_3, b_3, c_2, d_4)$. Estas dos tuplas tienen el mismo valor de C, c_2, pero tienen valores diferentes para A, a_2 y a_3, respectivamente. Por tanto, se ha hallado un par de tuplas t_1 y t_2 tal que $t_1[C] = t_2[C]$, pero $t_1[A] \neq t_2[A]$.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

Figura 8.4. Ejemplar de muestra de la relación r.

Se dice que algunas dependencias funcionales son **triviales** porque las satisfacen todas las relaciones. Por ejemplo, $A \rightarrow A$ la satisfacen todas las relaciones que implican al atributo A. La lectura literal de la definición de dependencia funcional deja ver que, para todas las tuplas t_1 y t_2 tal que $t_1[A] = t_2[A]$, se cumple que $t_1[A] = t_2[A]$. De manera análoga, $AB \rightarrow A$ la satisfacen todas las relaciones que implican al atributo A. En general, las dependencias funcionales de la forma $\alpha \rightarrow \beta$ son **triviales** si se cumple la condición $\beta \subseteq \alpha$.

1 Un profesor o un estudiante pueden estar asociados a más de un departamento, por ejemplo, una facultad asociada o un departamento menor. Nuestro esquema de universidad simplificada modela solamente el departamento principal al que está asociado un profesor o un estudiante. En una universidad real el esquema debería capturar las asociaciones secundarias en otras relaciones.

2 Tenga en cuenta que se supone que las relaciones son conjuntos. SQL puede tratar con multiconjuntos, y las declaraciones de claves primarias en SQL para un conjunto de atributos K requiere no solo que $t_1 = t_2$ si $t_1[K] = t_2[K]$, sino también que no haya tuplas duplicadas. SQL también requiere que a los atributos en el conjunto K no se les pueda asignar el valor *null*.

Es importante darse cuenta de que una relación dada puede, en cualquier momento, satisfacer algunas dependencias funcionales cuyo cumplimiento no sea necesario en el esquema de la relación. En la relación *aula* de la Figura 8.5 puede verse que se satisface *número_aula* → *capacidad*. Sin embargo, se sabe que en el mundo real dos aulas en diferentes edificios pueden tener el mismo número pero diferentes capacidades. Por tanto, es posible, en un momento dado, tener un ejemplar de la relación *aula* en el que no se satisfaga *número_aula* → *capacidad*. Por consiguiente, no se incluirá *número_aula* → *capacidad* en el conjunto de dependencias funcionales que se cumplen en la relación *aula*. Sin embargo, se espera que la dependencia funcional *edificio, número_aula* → *capacidad* se cumpla en el esquema *aula*.

Dado un conjunto de dependencias funcionales F que se cumple en una relación r (R), es posible inferir que también se deban cumplir en esa misma relación otras dependencias funcionales. Por ejemplo, dado el esquema r (A, B, C), si se cumplen en r las dependencias funcionales $A \rightarrow B$ y $B \rightarrow C$, se puede inferir que también $A \rightarrow C$ se debe cumplir en r. Esto es porque, dado cualquier valor de A, solo puede haber un valor correspondiente de B, y para ese valor de B, solo puede haber un valor correspondiente de C. Más adelante, en la Sección 8.4.1, se estudia la manera de realizar esas inferencias.

edificio	*número_aula*	*capacidad*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Figura 8.5. Ejemplar de la relación *aula*.

Se utiliza la notación F^+ para denotar el **cierre** del conjunto F, es decir, el conjunto de todas las dependencias funcionales que pueden inferirse dado el conjunto F. Evidentemente, F^+ contiene todas las dependencias funcionales de F.

8.3.2. Forma normal de Boyce–Codd

Una de las formas normales más deseables que se pueden obtener es la **forma normal de Boyce-Codd (FNBC).** Elimina todas las redundancias que se pueden descubrir a partir de las dependencias funcionales aunque, como se verá en la Sección 8.6, puede que queden otros tipos de redundancia. Un esquema de relación R está en la FNBC respecto al conjunto F de dependencias funcionales si, para todas las dependencias funcionales de F^+ de la forma $\alpha \rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple, al menos, una de las siguientes condiciones:

- $\alpha \rightarrow \beta$ es una dependencia funcional trivial (es decir, $\beta \subseteq \alpha$)
- α es superclave del esquema R.

Los diseños de bases de datos están en la FNBC si cada miembro del conjunto de esquemas de relación que constituye el diseño se halla en la FNBC.

Ya hemos visto en la Sección 8.1 un ejemplo de esquema relacional que no está en FNBC:

profesor_dept (*ID, nombre, sueldo, nombre_dept, edificio, presupuesto*)

La dependencia funcional *nombre_dept* → *presupuesto* se cumple en *profesor_dept*, pero *nombre_dept* no es una superclave, ya que cada departamento puede tener diferente número de profesores. En la Sección 8.1.2, se vio que la descomposición de *profesor_dept* en *profesor* y *departamento* conduce a un mejor diseño. El esquema *profesor* está en FNBC. Todas las dependencias funcionales no triviales, como:

ID → *nombre, nombre_dept, sueldo*

incluyen *ID* en la parte izquierda de la flecha, e *ID* es una superclave (realmente, en este caso, la clave primaria) para *profesor*. En otras palabras, existe una dependencia funcional no trivial con cualquier combinación de *nombre*, *nombre_dept* y *sueldo*, sin *ID*, en un lado. Por tanto, *profesor* está en FNBC.

De forma similar, el esquema *departamento* está en FNBC ya que todas las dependencias funcionales no triviales que se cumplen, como por ejemplo:

nombre_dept → *edificio, presupuesto*

incluyen *nombre_dept* en la parte izquierda de la flecha, y *nombre_dept* es una superclave (y clave primaria) de *departamento*. Por tanto, departamento está en FNBC.

Ahora se va a definir una regla general para la descomposición de esquemas que no se hallen en la FNBC. Sea R un esquema que no se halla en la FNBC. En ese caso, queda, al menos, una dependencia funcional no trivial $\alpha \rightarrow \beta$ tal que α no es superclave de R. En el diseño se sustituye R por dos esquemas:

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

En el caso anterior de *profesor_dept*, α = *nombre_dept*, β = {*edificio, presupuesto*}, y *profesor_dept*

se sustituye por:

- $(\alpha \cup \beta)$ = (*nombre_dept, edificio, presupuesto*)
- $(R - (\beta - \alpha))$ = (*ID, nombre, nombre_dept, sueldo*)

En este caso resulta que $\beta - \alpha = \beta$. Hay que definir la regla tal y como se ha hecho para que trate correctamente las dependencias funcionales que tienen atributos que aparecen a los dos lados de la flecha. Las razones técnicas para esto se tratarán más adelante, en la Sección 8.5.1.

Cuando se descomponen esquemas que no se hallan en la FNBC, puede que uno o varios de los esquemas resultantes no estén en FNBC. En ese caso, hacen falta más descomposiciones, cuyo resultado final sea un conjunto de esquemas en FNBC.

8.3.3. FNBC y la conservación de las dependencias

Se han visto varias maneras de expresar las restricciones de consistencia de las bases de datos: restricciones de clave primaria, dependencias funcionales, restricciones **check,** asertos y disparadores. La comprobación de estas restricciones cada vez que se actualiza la base de datos puede ser costosa y, por tanto, resulta útil diseñar la base de datos de forma que las restricciones se puedan comprobar de manera eficiente. En concreto, si la comprobación de las dependencias funcionales puede realizarse considerando solo una relación, el coste de comprobación de esa restricción será bajo. Se verá que la descomposición en la FNBC puede impedir la comprobación eficiente de determinadas dependencias funcionales.

Para ilustrar esto, supóngase que se lleva a cabo una modificación aparentemente pequeña en la organización de la universidad. En el diseño de la Figura 7.15 un estudiante solo puede tener un tutor. Esto se deduce del conjunto de relaciones *tutor* que es varios a uno de estudiante a *tutor*. El «pequeño» cambio que se va a hacer es que un profesor pueda estar asociado a un único departamento y un estudiante pueda tener más de un tutor, pero como mucho uno de un departamento dado.[3]

Una forma de implementar este cambio en el diseño E-R es sustituyendo la relación *tutor* por una relación ternaria, *tutor_dept*, que incluya tres conjuntos de entidades *profesor*, *estudiante* y *departamento* que es de varios a uno del par {*estudiante*, *profesor*} a *departamento*, como se muestra en la Figura 8.6. El diagrama E-R especifica la restricción de que «un estudiante puede tener más de un tutor, pero como mucho uno de un departamento dado».

3 Este tipo de relación tiene sentido en el caso de estudiantes que estudien dos carreras.

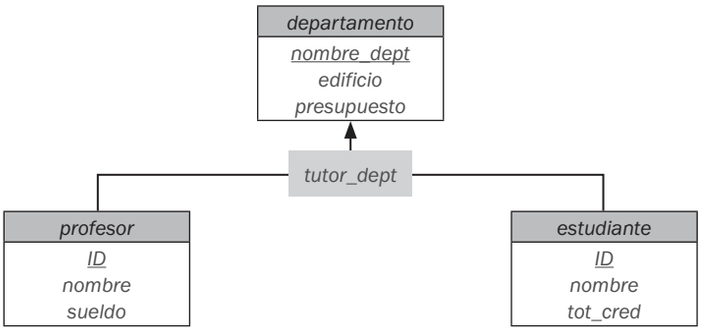


Figura 8.6. La relación *tutor_dept*.

Con este nuevo diagrama E-R, los esquemas de *profesor*, *departamento* y *estudiante* no cambian. Sin embargo, el esquema derivado de *tutor_dept* es ahora:

tutor_dept (*e_ID*, *p_ID*, *nombre_dept*)

Aunque no se especifique en el diagrama E-R, suponga que tenemos la restricción adicional de que «un profesor puede ser tutor solo en un único departamento».

Entonces, la siguiente dependencia funcional se cumple en *tutor_dept*:

$$p_ID \rightarrow nombre_dept$$
$$e_ID,\ nombre_dept \rightarrow p_ID$$

La primera dependencia funcional se sigue de nuestro requisito de que «un profesor puede ser tutor solo en un departamento». La segunda dependencia funcional se sigue de nuestro requisito de que «un estudiante puede tener como mucho un tutor de un departamento dado».

Tenga en cuenta que con este diseño, estamos obligados a repetir el nombre del departamento cada vez que un profesor participa en una relación *tutor_dept*. Se puede observar que *tutor_dept* no se encuentra en FNBC ya que *p_ID* no es una superclave. Siguiendo nuestra regla para la descomposición FNBC, se obtiene:

(*e_ID*, *p_ID*)
(*p_ID*, *nombre_dept*)

Ambos esquemas están en FNBC. De hecho, se puede verificar que cualquier esquema con solo dos atributos está en FNBC por definición. Tenga en cuenta, sin embargo, que en nuestro diseño FNBC no existe ningún esquema que incluya todos los atributos que aparecen en la dependencia funcional *e_ID, nombre_dept* → *p_ID*.

Como nuestro diseño hace que sea muy difícil computacionalmente forzar esta dependencia funcional, se dice que nuestro diseño no **preserva la dependencia.**[4] Como la preservación de la dependencia se suele considerar deseable, consideraremos otra forma normal, más débil que la FNBC, que nos permita preservar dependencias. A esta forma normal se le llama tercera forma normal.[5]

8.3.4. Tercera forma normal

La FNBC exige que todas las dependencias no triviales sean de la forma $\alpha \rightarrow \beta$, donde α es una superclave. La tercera forma normal (3FN) relaja ligeramente esta restricción al permitir dependencias funcionales no triviales cuya parte izquierda no sea una superclave. Antes de definir la 3FN hay que recordar que las claves candidatas son superclaves mínimas; es decir, superclaves de las que ningún subconjunto propio sea también superclave.

El esquema de relación R está **en tercera forma normal** respecto a un conjunto F de dependencias funcionales si, para todas las dependencias funcionales de F^+ de la forma $\alpha \rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple, al menos, una de las siguientes condiciones:

- $\alpha \rightarrow \beta$ es una dependencia funcional trivial.
- α es superclave de R.
- Cada atributo A de $\beta - \alpha$ está contenido en alguna clave candidata de R.

Obsérvese que la tercera condición no dice que una sola clave candidata deba contener todos los atributos de $\beta - \alpha$; cada atributo A de $\beta - \alpha$ puede estar contenido en una clave candidata *diferente*.

Las dos primeras alternativas son iguales que las dos alternativas de la definición de la FNBC. La tercera alternativa de la definición de la 3FN parece bastante poco intuitiva, y no resulta evidente el motivo de su utilidad. Representa, en cierto sentido, una relajación mínima de las condiciones de la FNBC que ayuda a garantizar que cada esquema tenga una descomposición que conserve las dependencias en la 3FN. Su finalidad se aclarará más adelante, cuando se estudie la descomposición en la 3FN

Obsérvese que cualquier esquema que satisfaga la FNBC satisface también la 3FN, ya que cada una de sus dependencias funcionales satisfará una de las dos primeras alternativas. Por tanto, la FNBC es una forma normal más restrictiva que la 3FN.

La definición de la 3FN permite ciertas dependencias funcionales que no se permiten en la FNBC. Las dependencias $\alpha \rightarrow \beta$ que solo satisfacen la tercera alternativa de la definición de la 3FN no se permiten en la FNBC, pero sí en la 3FN.[6]

Consideremos de nuevo la relación *tutor_dept*, que alberga las siguientes dependencias funcionales:

$$p_ID \rightarrow nombre_dept$$
$$e_ID,\ nombre_dept \rightarrow p_ID$$

En la Sección 8.3.3 se trató que la dependencia funcional «*p_ID* → *nombre_dept*» hacía que el esquema *tutor_dept* no estuviese en FNBC. Tenga en cuenta que aquí $\alpha = p_ID$, $\beta = nombre_dept$, y $\beta - \alpha = nombre_dept$. Como la dependencia funcional *e_ID, nombre_dept* →*p_ID* se cumple en *tutor_dept*, el atributo *nombre_dept* forma parte de una clave candidata y, por tanto, *tutor_dept* está en 3NF.

Se han abordado las contrapartidas que se pueden lograr entre FNBC y 3FN cuando el diseño FNBC no preserva dependencias. Estas contrapartidas se describen con más detalle en la Sección 8.5.4.

8.3.5. Formas normales superiores

Puede que en algunos casos el empleo de dependencias funcionales para la descomposición de los esquemas no sea suficiente para evitar la repetición innecesaria de información. Considérese una ligera variación de la definición del conjunto de entidades *profesor* en la que se registra con cada profesor un conjunto de nombres de niños y varios números de teléfono. Los números de teléfono pueden ser compartidos por varias personas. Entonces, *número_teléfono* y *nombre_niño* serán atributos multivalorados y, de acuerdo con las reglas para la generación de esquemas a partir de los diseños E-R, habrá dos esquemas, uno por cada uno de los atributos multivalorados *número_teléfono* y *nombre_niño*:

(*ID*, *nombre_niño*)
(*ID*, *número_teléfono*)

4 Técnicamente, es posible que una dependencia cuyos atributos no aparecen en ningún esquema se fuerce de forma implícita debido a la existencia de otras dependencias que la generan de forma lógica. Se tratará este caso más adelante en la Sección 8.4.5.

5 Se habrá dado cuenta de que nos hemos saltado la segunda forma normal. Solo tiene sentido desde un punto de vista histórico y en la práctica no se usa.

6 Estas dependencias son un ejemplo de **dependencias transitivas** (véase el Ejercicio práctico 8.16). La definición original de la 3FN se realizaba en términos de dependencias transitivas. La definición que utilizamos es equivalente pero más sencilla de entender.

Si se combinan estos dos esquemas para obtener:

(*ID, nombre_niño, número_teléfono*)

se descubre que el resultado se halla en la FNBC, ya que solo se cumplen dependencias funcionales no triviales. En consecuencia, se puede pensar que ese tipo de combinación es una buena idea. Sin embargo, se trata de una mala idea, como puede verse si se considera el ejemplo de un profesor con dos niños y dos números de teléfono. Por ejemplo, sea el profesor con ID 99999 que tiene dos niños llamados «David» y «William» y dos números de teléfono, 512-555-123 y 512-555-432. En el esquema combinado hay que repetir los números de teléfono una vez por cada dependiente:

(99999, David, 512-555-1234)
(99999, David, 512-555-4321)
(99999, William, 512-555-1234)
(99999, William, 512-555-4321)

Si no se repitieran los números de teléfono y solo se almacenaran la primera y la última tupla, se habrían guardado los nombres de los niños y los números de teléfono, pero las tuplas resultantes implicarían que David correspondería al 512-555-123 y que William correspondería al 512-555-432. Como sabemos, esto es incorrecto.

Debido a que las formas normales basadas en las dependencias funcionales no son suficientes para tratar con situaciones como esta, se han definido otras dependencias y formas normales. Se estudian en las Secciones 8.6 y 8.7.

8.4. Teoría de las dependencias funcionales

Se ha visto en los ejemplos que resulta útil poder razonar de manera sistemática sobre las dependencias funcionales como parte del proceso de comprobación de los esquemas para la FNBC o la 3FN.

8.4.1. Cierre de los conjuntos de dependencias funcionales

Se verá que, dado un conjunto F de dependencias funcionales, se puede probar que también se cumple alguna otra dependencia funcional. Se dice que F «implica lógicamente» esas dependencias funcionales. Cuando se verifican las formas normales, no es suficiente considerar el conjunto dado de dependencias funcionales. También hay que considerar *todas* las dependencias funcionales que se cumplen en el esquema.

De manera más formal, dado un esquema relacional $r(R)$, una dependencia funcional f de R está **implicada lógicamente** por un conjunto de dependencias funcionales F de r si cada ejemplar de la relación $r(R)$ que satisface F satisface también f.

Supóngase que se tiene el esquema de relación $r(A, B, C, G, H, I)$ y el conjunto de dependencias funcionales:

$$A \rightarrow B$$
$$A \rightarrow C$$
$$CG \rightarrow H$$
$$CG \rightarrow I$$
$$B \rightarrow H$$

La dependencia funcional:

$$A \rightarrow H$$

está implicada lógicamente. Es decir, se puede demostrar que siempre que el conjunto dado de dependencias funcionales se cumple en una relación, también se debe cumplir $A \rightarrow H$. Supóngase que t_1 y t_2 son tuplas tales que:

$$t_1[A] = t_2[A]$$

Como se tiene que $A \rightarrow B$, se deduce de la definición de dependencia funcional que:

$$t_1[B] = t_2[B]$$

Entonces, como se tiene que $B \rightarrow H$, se deduce de la definición de dependencia funcional que:

$$t_1[H] = t_2[H]$$

Por tanto, se ha demostrado que siempre que t_1 y t_2 sean tuplas tales que $t_1[A] = t_2[A]$, debe ocurrir que $t_1[H] = t_2[H]$. Pero esa es exactamente la definición de $A \rightarrow H$.

Sea F un conjunto de dependencias funcionales. El **cierre** de F, denotado por F^+, es el conjunto de todas las dependencias funcionales implicadas lógicamente por F. Dado F, se puede calcular F^+ directamente a partir de la definición formal de dependencia funcional. Si F fuera de gran tamaño, ese proceso sería largo y difícil. Este tipo de cálculo de F^+ requiere argumentos del tipo que se acaba de utilizar para demostrar que $A \rightarrow H$ está en el cierre del conjunto de dependencias de ejemplo.

Los **axiomas,** o reglas de inferencia, proporcionan una técnica más sencilla para el razonamiento sobre las dependencias funcionales. En las reglas que se ofrecen a continuación se utilizan las letras griegas ($\alpha, \beta, \gamma, \ldots$) para los conjuntos de atributos y las letras latinas mayúsculas desde el comienzo del alfabeto para los atributos. Se emplea $\alpha\beta$ para denotar $\alpha \cup \beta$.

Se pueden utilizar las tres reglas siguientes para hallar las dependencias funcionales implicadas lógicamente. Aplicando estas reglas *repetidamente* se puede hallar todo F^+, dado F. Este conjunto de reglas se denomina **axiomas de Armstrong** en honor a la persona que las propuso por primera vez.

- **Regla de la reflexividad.** Si α es un conjunto de atributos y $\beta \subseteq \alpha$, entonces se cumple que $\alpha \rightarrow \beta$.
- **Regla de la aumentatividad**. Si se cumple que $\alpha \rightarrow \beta$ y γ es un conjunto de atributos, entonces se cumple que $\gamma\alpha \rightarrow \gamma\beta$.
- **Regla de la transitividad**. Si se cumple que $\alpha \rightarrow \beta$ y que $\beta \rightarrow \gamma$, entonces se cumple que $\alpha \rightarrow \gamma$.

Los axiomas de Armstrong son **correctos,** ya que no generan dependencias funcionales incorrectas. Son **completos**, ya que, para un conjunto de dependencias funcionales F dado, permiten generar todo F^+. Las notas bibliográficas proporcionan referencias de las pruebas de su corrección y de su completitud.

Aunque los axiomas de Armstrong son completos, resulta pesado utilizarlos directamente para el cálculo de F^+. Para simplificar más las cosas se dan unas reglas adicionales. Se pueden utilizar los axiomas de Armstrong para probar que son correctas (véanse los Ejercicios prácticos 8.4, 8.5 y 8.26).

- **Regla de la unión.** Si se cumple que $\alpha \rightarrow \beta$ y que $\alpha \rightarrow \gamma$, entonces se cumple que $\alpha \rightarrow \beta\gamma$.
- **Regla de la descomposición.** Si se cumple que $\alpha \rightarrow \beta\gamma$, entonces se cumple que $\alpha \rightarrow \beta$ y que $\alpha \rightarrow \gamma$.
- **Regla de la pseudotransitividad.** Si se cumple que $\alpha \rightarrow \beta$ y que $\gamma\beta \rightarrow \delta$, entonces se cumple que $\alpha\gamma \rightarrow \delta$.

Se aplicarán ahora las reglas al ejemplo del esquema $R = (A, B, C, G, H, I)$ y del conjunto F de dependencias funcionales $\{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$. A continuación se relacionan varios miembros de F^+:

- $A \rightarrow H$. Dado que se cumplen $A \rightarrow B$ y $B \rightarrow H$, se aplica la regla de transitividad. Obsérvese que resultaba mucho más sencillo emplear los axiomas de Armstrong para demostrar que se cumple que $A \rightarrow H$, que deducirlo directamente a partir de las definiciones, como se ha hecho anteriormente en este apartado.
- $CG \rightarrow HI$. Dado que $CG \rightarrow H$ y $CG \rightarrow I$, la regla de la unión implica que $CG \rightarrow HI$.
- $AG \rightarrow I$. Dado que $A \rightarrow C$ y $CG \rightarrow I$, la regla de pseudotransitividad implica que se cumple que $AG \rightarrow I$.

Otra manera de averiguar que se cumple que $AG \rightarrow I$ es la siguiente: se utiliza la regla de aumentatividad en $A \rightarrow C$ para inferir que $AG \rightarrow CG$. Aplicando la regla de transitividad a esta dependencia y a $CG \rightarrow$ I, se infiere que $AG \rightarrow I$.

La Figura 8.7 muestra un procedimiento que demuestra formalmente el modo de utilizar los axiomas de Armstrong para calcular F^+. En este procedimiento puede que la dependencia funcional ya esté presente en F^+ cuando se le añade; en ese caso, no hay ninguna modificación en F^+. En la Sección 8.4.2 se verá una manera alternativa de calcular F^+.

Los términos a la derecha y a la izquierda de las dependencias funcionales son subconjuntos de R. Dado que un conjunto de tamaño n tiene 2^n subconjuntos, hay un total de 2^n 3 $2^n = 2^{2n}$ dependencias funcionales posibles, donde n es el número de atributos de R. Cada iteración del bucle del procedimiento repetido, salvo la última, añade como mínimo una dependencia funcional a F^+. Por tanto, está garantizado que el procedimiento termine.

8.4.2. Cierre del conjunto de atributos

Se dice que un atributo B está **determinado funcionalmente** por α si $\alpha \rightarrow$ B. Para comprobar si un conjunto α es superclave hay que diseñar un algoritmo para el cálculo del conjunto de atributos determinados funcionalmente por α. Una manera de hacerlo es calcular F^+, tomar todas las dependencias funcionales con α como término de la izquierda y tomar la unión de los términos de la derecha de todas esas dependencias. Sin embargo, hacer esto puede resultar costoso, ya que F^+ puede ser de gran tamaño.

Un algoritmo eficiente para el cálculo del conjunto de atributos determinados funcionalmente por α no solo resulta útil para comprobar si α es superclave, sino también para otras tareas, como se verá más adelante en este apartado.

```
F+ = F
repeat
   for each dependencia funcional f de F+
      aplicar las reglas de reflexividad y de aumentatividad a f
      añadir las dependencias funcionales resultantes a F+
   for each par de dependencias funcionales f1 y f2 de F+
      if f1 y f2 se pueden combinar mediante la transitividad
         añadir la dependencia funcional resultante a F+
until F+ deje de cambiar
```

Figura 8.7. Procedimiento para calcular F^+.

```
resultado := α;
repeat
   for each dependencia funcional β → γ in F do
      begin
         if β ⊆ resultado then resultado:= resultado ∪ γ;
      end
until (ya no cambie resultado)
```

Figura 8.8. Algoritmo para el cálculo de α^+, el cierre de α bajo F.

Sea α un conjunto de atributos. Al conjunto de todos los atributos determinados funcionalmente por α bajo un conjunto F de dependencias funcionales se le denomina **cierre** de α bajo F; se denota mediante α^+. La Figura 8.8 muestra un algoritmo, escrito en pseudocódigo, para calcular α^+. La entrada es un conjunto F de dependencias funcionales y el conjunto α de atributos. La salida se almacena en la variable *resultado*.

Para ilustrar el modo en que trabaja el algoritmo, se utilizará para calcular $(AG)^+$ con las dependencias funcionales definidas en la Sección 8.4.1. Se comienza con *resultado* $= AG$. La primera vez que se ejecuta el bucle **repeat** para comprobar cada dependencia funcional se halla que:

- $A \rightarrow B$ hace que se incluya B en resultado. Para comprobarlo, obsérvese que $A \rightarrow B$ se halla en F, y $A \subseteq$ resultado (que es AG), por lo que resultado:= resultado $\cup B$.
- $A \rightarrow C$ hace que *resultado* se transforme en $ABCG$.
- $CG \rightarrow H$ hace que *resultado* se transforme en $ABCGH$.
- $CG \rightarrow I$ hace que *resultado* se transforme en $ABCGHI$.

La segunda vez que se ejecuta el bucle **repeat** ya no se añade ningún atributo nuevo a *resultado*, y se termina el algoritmo.

Veamos ahora el motivo de que el algoritmo de la Figura 8.8 sea correcto. El primer paso es correcto, ya que $\alpha \rightarrow \alpha$ se cumple siempre (por la regla de reflexividad). Se afirma que, para cualquier subconjunto β de *resultado*, $\alpha \rightarrow \beta$. Dado que el bucle **repeat** se inicia con $\alpha \rightarrow$ *resultado* como cierto, solo se puede añadir γ a *resultado* si $\beta \subseteq$ *resultado* y $\beta \rightarrow \gamma$. Pero, entonces, *resultado* $\rightarrow \beta$ por la regla de reflexividad, por lo que $\alpha \rightarrow \beta$ por transitividad. Otra aplicación de la transitividad demuestra que $\alpha \rightarrow \gamma$ (empleando $\alpha \rightarrow \beta$ y $\beta \rightarrow \gamma$). La regla de la unión implica que $\alpha \rightarrow$ *resultado* $\cup \gamma$, por lo que α determina funcionalmente cualquier resultado nuevo generado en el bucle **repeat.** Por tanto, cualquier atributo devuelto por el algoritmo se halla en α^+.

Resulta sencillo ver que el algoritmo halla todo α^+. Si hay un atributo de α^+ que no se halle todavía en *resultado* en cualquier momento de la ejecución, debe haber una dependencia funcional $\beta \rightarrow \gamma$ para la que $\beta \subseteq$ resultado y, como mínimo, un atributo de γ no se halla en resultado. Cuando el algoritmo termina, se han procesado todas las dependencias funcionales, y se han añadido a *resultado* los atributos de γ; entonces podemos estar seguros de que todos los atributos de α^+ se encuentran en *resultado*.

En el peor de los casos, es posible que este algoritmo tarde un tiempo proporcional al cuadrado del tamaño de F. Hay un algoritmo más rápido (aunque ligeramente más complejo) que se ejecuta en un tiempo proporcional al tamaño de F; ese algoritmo se presenta como parte del Ejercicio práctico 8.8.

Existen varias aplicaciones del algoritmo de cierre de atributos:

- Para comprobar si α es superclave, se calcula α^+ y se comprueba si contiene todos los atributos de R.
- Se puede comprobar si se cumple la dependencia funcional $\alpha \rightarrow \beta$ (o, en otras palabras, si se halla en F^+), comprobando si $\beta \subseteq \alpha^+$. Es decir, se calcula α^+ empleando el cierre de los atributos y luego se comprueba si contiene a β. Esta prueba resulta especialmente útil, como se verá más adelante en este mismo capítulo.
- Ofrece una manera alternativa de calcular F^+: para cada $\gamma \subseteq R$ se halla el cierre γ^+, y para cada $S \subseteq \gamma^+$, se genera la dependencia funcional $\gamma \rightarrow S$.

8.4.3. Recubrimiento canónico

Supóngase que se tiene un conjunto F de dependencias funcionales de un esquema de relación. Siempre que un usuario lleve a cabo una actualización de la relación, el sistema de bases de datos debe asegurarse de que la actualización no viole ninguna dependencia funcional, es decir, que se satisfagan todas las dependencias funcionales de F en el nuevo estado de la base de datos.

El sistema debe retroceder la actualización si viola alguna dependencia funcional del conjunto F.

Se puede reducir el esfuerzo dedicado a la comprobación de las violaciones comprobando un conjunto simplificado de dependencias funcionales que tenga el mismo cierre que el conjunto dado. Cualquier base de datos que satisfaga el conjunto simplificado de dependencias funcionales satisfará también el conjunto original y viceversa, ya que los dos conjuntos tienen el mismo cierre. Sin embargo, el conjunto simplificado resulta más sencillo de comprobar.

En breve se verá el modo en que se puede crear ese conjunto simplificado. Antes, hacen falta algunas definiciones.

Se dice que un atributo de una dependencia funcional es **raro** si se puede eliminar sin modificar el cierre del conjunto de dependencias funcionales. La definición formal de los **atributos raros** es la siguiente: considérese un conjunto F de dependencias funcionales y la dependencia funcional $\alpha \rightarrow \beta$ de F.

- El atributo A es raro en α si $A \in \alpha$, y F implica lógicamente a $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
- El atributo A es raro en β si $A \in \beta$, y el conjunto de dependencias funcionales $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ implica lógicamente a F.

Por ejemplo, supóngase que se tienen las dependencias funcionales $AB \rightarrow C$ y $A \rightarrow C$ de F. Entonces, B es raro en $AB \rightarrow C$. Como ejemplo adicional, supóngase que se tienen las dependencias funcionales $AB \rightarrow CD$ y $A \rightarrow C$ de F. Entonces, C será raro en el lado derecho de $AB \rightarrow CD$.

Hay que tener cuidado con la dirección de las implicaciones al utilizar la definición de los atributos raros: si se intercambian el lado derecho y el izquierdo, la implicación se cumplirá *siempre*. Es decir, $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$ siempre implica lógicamente a F , y F también implica lógicamente siempre a $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$.

$F_c = F$
repeat
  Utilizar la regla de unión para sustituir las dependencias de F_c de la forma $\alpha_1 \rightarrow \beta_1$ y $\alpha_1 \rightarrow \beta_2$ con $\alpha_1 \rightarrow \beta_1 \beta_2$.
  Hallar una dependencia funcional $\alpha \rightarrow \beta$ de F_c con un atributo raro en α o en β.
    /* Nota: la comprobación de los atributos raros se lleva a cabo empleando F_c, no F */
  Si se halla algún atributo raro, hay que eliminarlo de $\alpha \rightarrow \beta$ en F_c.
until (F_c ya no cambie)

Figura 8.9. Cálculo del recubrimiento canónico.

A continuación se muestra el modo de comprobar de manera eficiente si un atributo es raro. Sea R el esquema de la relación y F el conjunto dado de dependencias funcionales que se cumplen en R. Considérese el atributo A de la dependencia $\alpha \rightarrow \beta$.

- Si $A \in \beta$, para comprobar si A es raro hay que considerar el conjunto:

$$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$$

  y comprobar si $\alpha \rightarrow A$ puede inferirse a partir de F'. Para ello hay que calcular α^+ (el cierre de α) bajo F'; si α^+ incluye a A, entonces A es raro en β.
- Si $A \in \alpha$, para comprobar si A es raro, sea $\gamma = \alpha - \{A\}$, hay que comprobar si se puede inferir que $\gamma \rightarrow \beta$ a partir de F. Para ello hay que calcular γ^+ (el cierre de γ) bajo F; si γ^+ incluye todos los atributos de β, entonces A es raro en α.

Por ejemplo, supóngase que F contiene $AB \rightarrow CD$, $A \rightarrow E$ y $E \rightarrow C$. Para comprobar si C es raro en $AB \rightarrow CD$, hay que calcular el cierre de los atributos de AB bajo $F' = \{AB \rightarrow D, A \rightarrow E$ y $E \rightarrow C\}$. El cierre es $ABCDE$, que incluye a CD, por lo que se infiere que C es raro.

El **recubrimiento canónico** F_c de F es un conjunto de dependencias tal que F implica lógicamente todas las dependencias de F_c y F_c implica lógicamente todas las dependencias de F. Además, F_c debe tener las propiedades siguientes:

- Ninguna dependencia funcional de F_c contiene atributos raros.
- El lado izquierdo de cada dependencia funcional de F_c es único. Es decir, no hay dos dependencias $\alpha_1 \rightarrow \beta_1$ y $\alpha_2 \rightarrow \beta_2$ de F_c tales que $\alpha_1 = \alpha_2$.

El recubrimiento canónico del conjunto de dependencias funcionales F puede calcularse como se muestra en la Figura 8.9. Es importante destacar que, cuando se comprueba si un atributo es raro, la comprobación utiliza las dependencias del valor actual de F_c, y **no** las dependencias de F. Si una dependencia funcional solo contiene un atributo en su lado derecho, por ejemplo, $A \rightarrow C$, y se descubre que ese atributo es raro, se obtiene una dependencia funcional con el lado derecho vacío. Hay que eliminar esas dependencias funcionales.

Se puede demostrar que el recubrimiento canónico de F, F_c, tiene el mismo cierre que F; por tanto, comprobar si se satisface F_c es equivalente a comprobar si se satisface F. Sin embargo, F_c es mínimo en un cierto sentido; no contiene atributos raros, y combina las dependencias funcionales con el mismo lado izquierdo. Resulta más económico comprobar F_c que comprobar el propio F.

Considérese el siguiente conjunto F de dependencias funcionales para el esquema (A, B, C):

$$A \rightarrow BC$$
$$B \rightarrow C$$
$$A \rightarrow B$$
$$AB \rightarrow C$$

Calcúlese el recubrimiento canónico de F.

- Hay dos dependencias funcionales con el mismo conjunto de atributos a la izquierda de la flecha:

$$A \rightarrow BC$$
$$A \rightarrow B$$

  Estas dependencias funcionales se combinan en $A \rightarrow BC$.
- A es raro en $AB \rightarrow C$, ya que F implica lógicamente a $(F - \{AB \rightarrow C\}) \cup \{B \rightarrow C\}$. Esta aseveración es cierta porque $B \rightarrow C$ ya se halla en el conjunto de dependencias funcionales.
- C es raro en $A \rightarrow BC$, ya que $A \rightarrow BC$ está implicada lógicamente por $A \rightarrow B$ y $B \rightarrow C$.

  Por tanto, su recubrimiento canónico es:

$$A \rightarrow B$$
$$B \rightarrow C$$

Dado un conjunto F de dependencias funcionales, puede suceder que toda una dependencia funcional del conjunto sea rara, en el sentido de que eliminarla no modifique el cierre de F. Se puede demostrar que el recubrimiento canónico F_c de F no contiene esa dependencia funcional rara. Supóngase que, por el contrario, esa dependencia rara estuviera en F_c. Los atributos del lado derecho de la dependencia serían raros, lo que no es posible por la definición de recubrimiento canónico.

Puede que el recubrimiento canónico no sea único. Por ejemplo, considérese el conjunto de dependencias funcionales $F = \{A \rightarrow BC, B \rightarrow AC$ y $C \rightarrow AB\}$. Si se aplica la prueba de rareza a $A \rightarrow BC$ se descubre que tanto B como C son raros bajo F. Sin embargo, sería incorrecto eliminar los dos. El algoritmo para hallar el recubrimiento canónico selecciona uno de los dos y lo elimina. Entonces:

1. Si se elimina C, se obtiene el conjunto $F' = \{A \rightarrow B, B \rightarrow AC$ y $C \rightarrow AB\}$. Ahora B ya no es raro en el lado de $A \rightarrow B$ bajo F'. Siguiendo con el algoritmo se descubre que A y B son raros en el lado derecho de $C \rightarrow AB$, lo que genera dos recubrimientos canónicos:

$$F_c = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$$
$$F_c = \{A \rightarrow B, B \rightarrow AC, C \rightarrow B\}$$

2. Si se elimina B, se obtiene el conjunto $\{A \rightarrow C, B \rightarrow AC$ y $C \rightarrow AB\}$. Este caso es simétrico del anterior y genera dos recubrimientos canónicos:

$$F_c = \{A \rightarrow C, C \rightarrow B, \text{ y } B \rightarrow A\}$$
$$F_c = \{A \rightarrow C, B \rightarrow C, \text{ y } C \rightarrow AB\}$$

Como ejercicio, el lector debe intentar hallar otro recubrimiento canónico de F.

8.4.4. Descomposición sin pérdidas

Sea $r(R)$ un esquema de relación y F un conjunto de dependencias funcionales de $r(R)$. Supóngase que R_1 y R_2 forman una descomposición de R. Se dice que la descomposición es una **descomposición sin pérdidas** si no hay pérdida de información al sustituir $r(R)$ por dos esquemas de relación $r_1(R_1)$ y $r_2(R_2)$. De forma más precisa, se dice que la descomposición es sin pérdidas si para todos los ejemplares legales de la base de datos (es decir, los ejemplares de la base de datos que satisfacen las dependencias funcionales especificadas y otras restricciones), la relación r contiene el mismo conjunto de tuplas como resultado de la siguiente consulta SQL:

```
select *
from (select R1 from r)
  natural join
  (select R2 from r)
```

Lo que se puede expresar de forma más sucinta en álgebra relacional como:

$$\pi_{R_1}(r) \bowtie \pi_{R_2}(r) = r$$

En otros términos, si se proyecta r sobre R_1 y R_2 y se calcula la reunión natural del resultado de la proyección, se vuelve a obtener exactamente r. Las descomposiciones que no son sin pérdidas se denominan **descomposiciones con pérdidas.** Los términos **descomposición de reunión sin pérdidas** y **descomposición de reunión con pérdidas** se utilizan a veces en lugar de descomposición sin pérdidas y descomposición con pérdidas.

A modo de ejemplo de una descomposición con pérdidas, recuerde la descomposición del esquema *empleado* en:

empleado1 (*ID, nombre*)
empleado2 (*nombre, calle, ciudad, sueldo*)

que ya se vio en la Sección 8.1.2. Como se ha visto en la Figura 8.3, el resultado de *empleado1* $\bowtie$ *empleado2* es un superconjunto de la relación original *empleado*, pero la descomposición es sin pérdidas, pues el resultado de la unión ha perdido información sobre qué identificador de empleado corresponde con qué direcciones y sueldos, en el caso de que dos o más empleados tengan el mismo nombre.

Se pueden utilizar las dependencias funcionales para probar si ciertas descomposiciones son sin pérdidas. Sean R, R_1, R_2 y F como se acaban de definir. R_1 y R_2 forman una descomposición sin pérdidas de R si, como mínimo, una de las dependencias funcionales siguientes se halla en F^+:

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$

En otros términos, si $R_1 \cap R_2$ forma una superclave de R_1 o de R_2, la descomposición de R es una descomposición sin pérdidas. Se puede utilizar el cierre de los atributos para buscar superclaves de manera eficiente, como ya se ha visto antes.

Para ilustrar esto, considérese el esquema

profesor_dept (*ID, nombre, sueldo, nombre_dept, edificio, presupuesto*)

cuya descomposición se hizo en la Sección 8.1.2 en los esquemas *profesor* y *departamento*:

profesor (*ID, nombre, nombre_dept, sueldo*)
departamento (*nombre_dept, edificio, presupuesto*)

Considérese la intersección de estos dos esquemas, que es *nombre_dept*. Lo vemos porque *nombre_dept* → *nombre_dept, edificio, presupuesto*, satisface la regla de descomposición sin pérdidas.

Para el caso general de la descomposición simultánea de un esquema en varios, la comprobación de la descomposición sin pérdidas es más complicada. Véanse las notas bibliográficas para tener referencias sobre este tema.

Mientras que la comprobación de la descomposición binaria es, claramente, una condición suficiente para la descomposición sin pérdidas, solo se trata de una condición necesaria si todas las restricciones son dependencias funcionales. Más adelante se verán otros tipos de restricciones (especialmente, un tipo de restricción denominada dependencia multivalorada, que se estudia en la Sección 8.6.1), que pueden garantizar que una descomposición sea sin pérdidas aunque no esté presente ninguna dependencia funcional.

8.4.5. Conservación de las dependencias

Resulta más sencillo caracterizar la conservación de las dependencias empleando la teoría de las dependencias funcionales que mediante el enfoque ad hoc que se utilizó en la Sección 8.3.3.

Sea F un conjunto de dependencias funcionales del esquema R y $R_1, R_2 \ldots, R_n$ una descomposición de R. La **restricción** de F a R_i es el conjunto F_i de todas las dependencias funcionales de F^+ que *solo* incluyen atributos de R_i. Dado que todas las dependencias funcionales de cada restricción implican a los atributos de un único esquema de relación, es posible comprobar el cumplimiento de esas dependencias verificando solo una relación.

Obsérvese que la definición de restricción utiliza todas las dependencias de F^+, no solo las de F. Por ejemplo, supóngase que se tiene $F = \{A \rightarrow B, B \rightarrow C\}$ y que se tiene una descomposición en AC y AB. La restricción de F a AC es, por tanto $A \rightarrow C$, ya que $A \rightarrow C$ se halla en F^+, aunque no se halle en F.

El conjunto de restricciones $F_1, F_2 \ldots, F_n$ es el conjunto de dependencias que pueden comprobarse de manera eficiente. Ahora cabe preguntarse si es suficiente comprobar solo las restricciones. Sea $F' = F_1 \cup F_2 \cup \ldots \cup F_n$. F' es un conjunto de dependencias funcionales del esquema R pero, en general $F' \neq F$. Sin embargo, aunque $F' \neq F$, puede ocurrir que $F'^+ = F^+$. Si esto último es cierto, entonces, todas las dependencias de F están implicadas lógicamente por F' y, si se comprueba que se satisface F', se habrá comprobado que se satisface F. Se dice que las descomposiciones que tienen la propiedad $F'^+ = F^+$ son **descomposiciones que conservan las dependencias.**

La Figura 8.10 muestra un algoritmo para la comprobación de la conservación de las dependencias. La entrada es el conjunto $D = \{R_1, R_2 \ldots, R_n\}$ de esquemas de relaciones descompuestas y el conjunto F de dependencias funcionales. Este algoritmo resulta costoso, ya que exige el cálculo de F^+. En lugar de aplicar el algoritmo de la Figura 8.10, se consideran dos alternativas.

```
calcular F+;
for each esquema Ri in D do
   begin
      Fi : = la restricción de F+ a Ri;
   end
F' : = ∅
for each restricción Fi do
   begin
      F' = F' ∪ Fi
   end
calcular F'+;
if ( F'+ = F+) then return (cierto)
   else return (falso);
```

Figura 8.10. Comprobación de la conservación de las dependencias.

En primer lugar, hay que tener en cuenta que si se puede comprobar cada miembro de F en una de las relaciones de la descomposición, la descomposición conserva las dependencias. Se trata de una manera sencilla de demostrar la conservación de las dependencias; no obstante, no funciona siempre. Hay casos en los que, aunque la descomposición conserve las dependencias, hay alguna dependencia de F que no se puede comprobar para ninguna relación

de la descomposición. Por tanto, esta prueba alternativa solo se puede utilizar como condición suficiente que es fácil de comprobar; si falla, no se puede concluir que la descomposición no conserve las dependencias; en vez de eso, hay que aplicar la prueba general.

A continuación se da una comprobación alternativa de la conservación de las dependencias, que evita tener que calcular F^+. La idea intuitiva subyacente a esta comprobación se explicará tras presentarla. La comprobación aplica el procedimiento siguiente a cada $\alpha \rightarrow \beta$ de F.

resultado $= \alpha$
repeat
  for each R_i de la descomposición
    $t = (resultado \cap R_i)^+ \cap R_i$
    $resultado = resultado \cup t$
until (no haya cambios en *resultado*)

En este caso, el cierre de los atributos se halla bajo el conjunto de dependencias funcionales F. Si *resultado* contiene todos los atributos de β, se conserva la dependencia funcional $\alpha \rightarrow \beta$. La descomposición conserva las dependencias si, y solo si, el procedimiento prueba que se conservan todas las dependencias de F.

Las dos ideas claves subyacentes a la comprobación anterior son las siguientes:

- La primera idea es comprobar cada dependencia funcional $\alpha \rightarrow \beta$ de F para ver si se conserva en F' (donde F' es tal y como se define en la Figura 8.10). Para ello, se calcula el cierre de α bajo F'; la dependencia se conserva exactamente cuando el cierre incluye a β. La descomposición conserva la dependencia si, y solo si, se comprueba que se conservan todas las dependencias de F.
- La segunda idea es emplear una forma modificada del algoritmo de cierre de atributos para calcular el cierre bajo F', sin llegar a calcular antes F'. Se desea evitar el cálculo de F', ya que resulta bastante costoso. Téngase en cuenta que F' es la unión de las F_i, donde F_i es la restricción de F sobre R_i. El algoritmo calcula el cierre de los atributos de ($resultado \cap R_i$) con respecto a F, intersecta el cierre con R_i y añade el conjunto de atributos resultante a *resultado*; esta secuencia de pasos es equivalente al cálculo del cierre de resultado bajo F_i. La repetición de este paso para cada i del interior del bucle genera el cierre de resultado bajo F'.

  Para comprender el motivo de que este enfoque modificado al cierre de atributos funcione correctamente, hay que tener en cuenta que para cualquier $\gamma \subseteq R_i$, $\gamma \rightarrow \gamma^+$ es una dependencia funcional en F^+, y $\gamma \rightarrow \gamma^+ \cap R_i$ es una dependencia funcional que se halla en F_i, la restricción de F^+ a R_i. A la inversa, si $\gamma \rightarrow \delta$ estuvieran en F_i, δ sería un subconjunto de $\gamma^+ \cap R_i$.

Esta comprobación tarda un tiempo que es polinómico, en lugar del exponencial necesario para calcular F^+.

8.5. Algoritmos de descomposición

Los esquemas de bases de datos del mundo real son mucho mayores que los ejemplos que caben en las páginas de un libro. Por este motivo, hacen falta algoritmos para la generación de diseños que se hallen en la forma normal adecuada. En este apartado se presentan algoritmos para la FNBC y para la 3FN.

8.5.1. Descomposición en la FNBC

Se puede emplear la definición de la FNBC para comprobar directamente si una relación se halla en esa forma normal. Sin embargo, el cálculo de F^+ puede resultar una tarea tediosa. En primer lugar se van a describir pruebas simplificadas para verificar si una relación dada se halla en la FNBC. En caso de que no lo esté, se puede descomponer para crear relaciones que sí estén en la FNBC. Más avanzado este apartado, se describirá un algoritmo para crear descomposiciones sin pérdidas de las relaciones, de modo que esas descomposiciones se hallen en la FNBC.

8.5.1.1. Comprobación de la FNBC

La comprobación de un esquema de relación R para ver si satisface la FNBC se puede simplificar en algunos casos:

- Para comprobar si la dependencia no trivial $\alpha \rightarrow \beta$ provoca alguna violación de la FNBC hay que calcular α^+ (el cierre de los atributos de α), y comprobar si incluye todos los atributos de R; es decir, si es superclave de R.
- Para comprobar si el esquema de relación R se halla en la FNBC basta con comprobar solo si las dependencias del conjunto F dado violan la FNBC, en vez de comprobar todas las dependencias de F^+.

  Se puede probar que, si ninguna de las dependencias de F provoca violaciones de la FNBC, ninguna de las dependencias de F^+ lo hará tampoco.

Por desgracia, el último procedimiento no funciona cuando se descomponen relaciones. Es decir, en las descomposiciones *no* basta con emplear F cuando se comprueba una relación R_i, en una descomposición de R, para la violación de la FNBC. Por ejemplo, considérese el esquema de relación R (A, B, C, D, E), con las dependencias funcionales F que contienen $A \rightarrow B$ y $BC \rightarrow D$. Supóngase que se descompusiera en $R_1(A, B)$ y en $R_2(A, C, D, E)$. Ahora ninguna de las dependencias de F contiene únicamente atributos de (A, C, D, E), por lo que se podría creer erróneamente que R_2 satisface la FNBC. De hecho, hay una dependencia $AC \rightarrow D$ de F^+ (que se puede inferir empleando la regla de la pseudotransitividad con las dos dependencias de F), que prueba que R_2 no se halla en la FNBC. Por tanto, puede que haga falta una dependencia que esté en F^+, pero no en F, para probar que una relación descompuesta no se halla en la FNBC.

Una comprobación alternativa de la FNBC resulta a veces más sencilla que calcular todas las dependencias de F^+. Para comprobar si una relación R_i de una descomposición de R se halla en la FNBC, se aplica esta comprobación:

- Para cada subconjunto α de atributos de R_i se comprueba que α^+ (el cierre de los atributos de α bajo F) no incluye ningún atributo de $R_i - \alpha$, o bien incluye todos los atributos de R_i.

```
resultado := {R};
hecho := falso;
calcular F+;
while (not hecho) do
  if (hay algún esquema Ri en resultado que no se halle en la FNBC)
    then begin
      sea α → β una dependencia funcional no trivial que se cumple
      en Ri tal que α → Ri no se halla en F+, y α ∩ β = ∅;
      resultado: = (resultado − Ri) ∪ (Ri − β) ∪ ( α, β);
  end
  else hecho: = cierto;
```

Figura 8.11. Algoritmo de descomposición en la FNBC.

Si algún conjunto de atributos α de R_i viola esta condición, considérese la siguiente dependencia funcional, que se puede probar que se halla en F^+:

$$\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i.$$

La dependencia anterior muestra que R_i viola la FNBC.

8.5.1.2. Algoritmo de descomposición de la FNBC

Ahora se puede exponer un método general para descomponer los esquemas de relación de manera que satisfagan la FNBC. La Figura 8.11 muestra un algoritmo para esta tarea. Si R no está en la FNBC, se puede descomponer R en un conjunto de esquemas en la FNBC, $R_1, R_2 \ldots, R_n$ utilizando este algoritmo. El algoritmo utiliza las dependencias que demuestran la violación de la FNBC para llevar a cabo la descomposición.

La descomposición que genera este algoritmo no solo está en la FNBC, sino que también es una descomposición sin pérdidas. Para ver el motivo por el que el algoritmo solo genera descomposiciones sin pérdidas hay que darse cuenta de que, cuando se sustituye el esquema R_i por $(R_i - \beta)$ y (α, β), se cumple que $\alpha \rightarrow \beta$ y que $(R_i - \beta) \cap (\alpha, \beta) = \alpha$.

Si no se exigiera que $\alpha \cap \beta = \varnothing$, los atributos de $\alpha \cap \beta$ no aparecerían en el esquema $((R_i - \beta)$, y ya no se cumpliría la dependencia $\alpha \rightarrow \beta$.

Es fácil ver que la descomposición de *profesor_dept* de la Sección 8.3.2 se obtiene de la aplicación del algoritmo. La dependencia funcional *nombre_dept* → *edificio*, *presupuesto* satisface la condición $\alpha \cap \beta = \varnothing$ y, por tanto, se elige para descomponer el esquema.

El algoritmo de descomposición en la FNBC tarda un tiempo exponencial en relación con el tamaño del esquema inicial, ya que el algoritmo, para comprobar si las relaciones de la descomposición satisfacen la FNBC, puede tardar un tiempo exponencial. Las notas bibliográficas ofrecen referencias de un algoritmo que puede calcular la descomposición en la FNBC en un tiempo polinómico. Sin embargo, puede que ese algoritmo «sobrenormalice», es decir, descomponga relaciones sin que sea necesario.

A modo de ejemplo, más prolijo, del empleo del algoritmo de descomposición en la FNBC, supóngase que se tiene un diseño de base de datos que emplea el siguiente esquema *clase*:

clase (*asignatura_id, nombre_asig, nombre_dept, créditos, secc_id, semestre, año, edificio, número_aula, capacidad, franja_horaria_id*)

El conjunto de dependencias funcionales a exigir que se cumplan en *clase* es:

asignatura_id → *nombre_asig, nombre_dept, créditos,*
edificio, número_aula → *capacidad asignatura_id, secc_id,*
semestre, año → *edificio, número_aula, franja_horaria_id*

Una clave candidata para este esquema es {*asignatura_id, secc_id, semestre, año*}.

Se puede aplicar el algoritmo de la Figura 8.11 al ejemplo *clase* de la manera siguiente:

- La dependencia funcional:

  asignatura_id → *nombre_asig, nombre_dept, créditos*

  se cumple, pero *asignatura_id* no es superclave. Por tanto, *clase* no se halla en FNBC. Se sustituye *clase* por:

asignatura (*asignatura_id, nombre_asig, nombre_dept, créditos*)
clase-1 (*asignatura_id, secc_id, semestre, año, edificio,*
número_aula capacidad, franja_horaria_id)

Entre las pocas dependencias funcionales no triviales que se cumplen en *asignatura* está *asignatura_id*, al lado izquierdo de la flecha. Dado que *asignatura_id* es clave de *asignatura*, la relación *asignatura* se halla en la FNBC.

- Una clave candidata para *clase-1* es {*asignatura_id, secc_id, semestre, año*}. La dependencia funcional:

  edificio, número_aula → *capacidad*

  se cumple en *clase-1*, pero {*edificio, número_aula*} no es superclave de *clase-1*. Se sustituye *clase-1* por:

aula (*edificio, número_aula, capacidad*)
sección (*asignatura_id, secc_id, semestre, año,*
edificio, número_aula, franja_horaria_id)

aula y *sección* se hallan en FNBC.

Por tanto, la descomposición de *clase* da lugar a los tres esquemas de relación *asignatura*, *aula* y *sección*, cada uno de los cuales se halla en la FNBC. Se corresponden con esquemas ya usados en este y en capítulos anteriores. Se puede comprobar que la descomposición es sin pérdidas y conserva las dependencias.

```
sea F_c un recubrimiento canónico de F;
i := 0;
for each dependencia funcional α → β de F_c
   i := i + 1;
   R_i := α β;
if ninguno de los esquemas R_j, j = 1, 2… , i contiene una clave candidata
de R
   then
      i := i + 1;
      R_i := cualquier clave candidata de R;
/* Opcionalmente, eliminar las relaciones redundantes */
repeat
   if algún esquema R_j se encuentra contenido en otro esquema R_k
      then
         /* Eliminar R_j */
         R_j := R_i;
         i: = i – 1;
until no se puedan eliminar más R_j
return (R_1, R_2… , R_i)
```

Figura 8.12. Comprobación de la conservación de las dependencias.

8.5.2. Descomposición en la 3FN

La Figura 8.12 muestra un algoritmo para la búsqueda de descomposiciones en la 3FN sin pérdidas y que conserven las dependencias. El conjunto de dependencias F_c utilizado en el algoritmo es un recubrimiento canónico de F. Obsérvese que el algoritmo considera el conjunto de esquemas $R_j, j = 1, 2 \ldots, i$; inicialmente $i = 0$ y, en ese caso, el conjunto está vacío.

Se aplicará este algoritmo al ejemplo de la Sección 8.3.4 en el que se demostró que:

tutor_dept (*e_ID, p_ID, nombre_dept*)

se halla en la 3FN, aunque no se halle en la FNBC. El algoritmo utiliza las dependencias funcionales de F:

f_1: *p_ID* → *nombre_dept*
f_2: *e_ID, nombre_dept* → *p_ID*

No existen atributos extraños en ninguna de las dependencias funcionales de F, por lo que F_c contiene f_1 y f_2. El algoritmo genera como R_1 el esquema (*p_ID nombre_dept*), y como R_2 el esquema (*e_ID, nombre_dept, p_ID*). El algoritmo, entonces, descubre que R_2 contiene una clave candidata, por lo que no se crean más esquemas de relación.

El conjunto de esquemas que resulta puede contener esquemas redundantes, sin un esquema R_k que contenga todos los atributos de otro esquema R_j. Por ejemplo, el esquema R_2 anterior contiene todos los atributos de R_1. El algoritmo elimina todos estos esquemas que ya están contenidos en otro esquema. Cualquier dependencia que se pueda comprobar en R_j y sea eliminada también se puede comprobar en la correspondiente relación R_k, y la descomposición es sin pérdidas aunque se elimine R_j.

Ahora, consideremos de nuevo el esquema *clase* de la Sección 8.5.1.2 y apliquemos el algoritmo de descomposición de 3FN. El conjunto de dependencias funcionales que se listan resultan ser un recubrimiento canónico. Como resultado, el algoritmo nos da los mismos tres esquemas *asignatura*, *aula* y *sección*.

El ejemplo anterior muestra una propiedad interesante del algoritmo de 3FN. A veces, el resultado no solo está en 3FN, sino también en FNBC. Ello sugiere un método alternativo de generar un diseño en FNBC. En primer lugar se utiliza el algoritmo de 3FN. Después, para los esquemas del diseño en 3FN que no estén en FNBC, se descomponen utilizando el algoritmo de FNBC. Si el resultado no preserva las dependencias, se vuelve al diseño en 3FN.

8.5.3. Corrección del algoritmo de 3FN

El algoritmo garantiza la conservación de las dependencias mediante la creación explícita de un esquema para cada dependencia del recubrimiento canónico. Asegura que la descomposición sea sin pérdidas al garantizar que, como mínimo, un esquema contenga una clave candidata del esquema que se está descomponiendo. El Ejercicio práctico 8.15 proporciona algunos indicios de la prueba de que esto basta para garantizar una descomposición sin pérdidas.

Este algoritmo también se denomina **algoritmo de síntesis de la 3FN,** ya que toma un conjunto de dependencias y añade los esquemas de uno en uno, en lugar de descomponer el esquema inicial de manera repetida. El resultado no queda definido de manera única, ya que cada conjunto de dependencias funcionales puede tener más de un recubrimiento canónico y, además, en algunos casos, el resultado del algoritmo depende del orden en que considere las dependencias en F_c. El algoritmo puede descomponer una relación incluso si ya está en 3FN, sin embargo, se garantiza que la descomposición también está en 3FN.

Si una relación R_i está en la descomposición generada por el algoritmo de síntesis, entonces R_i está en la 3FN. Recuérdese que, cuando se busca la 3FN, basta con considerar las dependencias funcionales cuyo lado derecho sea un solo atributo. Por tanto, para ver si R_i está en la 3FN, hay que convencerse de que cualquier dependencia funcional $\gamma \rightarrow B$ que se cumpla en R_i satisface la definición de la 3FN. Supóngase que la dependencia que generó R_i en el algoritmo de síntesis es $\alpha \rightarrow \beta$. Ahora bien, B debe estar en α o en β, ya que B está en R_i y $\alpha \rightarrow \beta$ ha generado R_i. Considérense los tres casos posibles:

- B está tanto en α como en β. En ese caso, la dependencia $\alpha \rightarrow \beta$ no habría estado en F_c, ya que B sería rara en β. Por tanto, este caso no puede darse.
- B está en β pero no en α. Considérense dos casos:
  - γ es superclave. Se satisface la segunda condición de la 3FN.
  - γ no es superclave. Entonces, α debe contener algún atributo que no se halle en γ. Ahora bien, como $\gamma \rightarrow$ B se halla en F^+, debe poder obtenerse a partir de F_c mediante el algoritmo del cierre de atributos de γ. La obtención no puede haber empleado $\alpha \rightarrow \beta$; si lo hubiera hecho, α debería estar contenida en el cierre de los atributos de γ, lo que no es posible, ya que se ha dado por supuesto que γ no es superclave. Ahora bien, empleando $\alpha \rightarrow (\beta - \{B\})$ y $\gamma \rightarrow$ B, se puede obtener que $\alpha \rightarrow$ B (puesto que $\gamma \subseteq \alpha\beta$ y γ no puede contener a B porque $\gamma \rightarrow B$ no es trivial). Esto implicaría que B es raro en el lado derecho de $\alpha \rightarrow \beta$, lo que no es posible, ya que $\alpha \rightarrow \beta$ está en el recubrimiento canónico F_c. Por tanto, si B está en β, γ debe ser superclave, y se debe satisfacer la segunda condición de la 3FN.
- B está en α pero no en β.

  Como α es clave candidata, se satisface la tercera alternativa de la definición de la 3FN.

Es interesante que el algoritmo que se ha descrito para la descomposición en la 3FN pueda implementarse en tiempo polinómico, aunque la comprobación de una relación dada para ver si satisface la 3FN sea NP-dura (lo que hace muy improbable que se invente nunca un algoritmo de tiempo polinómico para esta tarea).

8.5.4. Comparación entre la FNBC y la 3FN

De las dos formas normales para los esquemas de las bases de datos relacionales, la 3FN y la FNBC, la 3FN es más conveniente, ya que siempre es posible obtener un diseño en la 3FN sin sacrificar la ausencia de pérdidas ni la conservación de las dependencias. Sin embargo, la 3FN presenta inconvenientes: puede que haya que emplear valores nulos para representar algunas de las relaciones significativas posibles entre los datos, y existe el problema de la repetición de la información.

Los objetivos del diseño de bases de datos con dependencias funcionales son:

1. FNBC.
2. Ausencia de pérdidas.
3. Conservación de las dependencias.

Como no siempre resulta posible satisfacer las tres, puede que nos veamos obligados a escoger entre la FNBC y la conservación de las dependencias con la 3FN.

Merece la pena destacar que el SQL no ofrece una manera de especificar las dependencias funcionales, salvo para el caso especial de la declaración de las superclaves mediante las restricciones **primary key** o **unique.** Es posible, aunque un poco complicado, escribir asertos que hagan que se cumpla una dependencia funcional determinada (véase el Ejercicio práctico 8.9); por desgracia, no existe actualmente ningún sistema de base de datos que disponga de los complejos asertos que serían necesarios para forzar las dependencias funcionales, y la comprobación de los asertos resultaría muy costosa. Por tanto, aunque se tenga una descomposición que conserve las dependencias, si se utiliza el SQL estándar, solo se podrán comprobar de manera eficiente las dependencias funcionales cuyo lado izquierdo sea una clave.

Aunque puede que la comprobación de las dependencias funcionales implique una operación reunión si la descomposición no conserva las dependencias, se puede reducir su coste empleando vistas materializadas, utilizables en la mayor parte de los sistemas de bases de datos. Dada una descomposición en la FNBC que no conserve las dependencias, se considera cada dependencia en un recubrimiento canónico F_c que no se conserve en la descomposición. Para cada una de esas dependencias $\alpha \rightarrow \beta$, se define una vista materializada que calcule una reunión de todas las relaciones de la descomposición y proyecte el resultado sobre $\alpha\beta$. La dependencia funcional puede comprobarse fácilmente en la vista materializada mediante una restricción **unique** (α) o **primary key** (α).

La parte negativa es que hay una sobrecarga espacial y temporal debida a la vista materializada, pero la positiva es que el programador de la aplicación no tiene que preocuparse de escribir código para mantener los datos redundantes consistentes en las actualizaciones; es labor del sistema de bases de datos conservar la vista materializada, es decir, mantenerla actualizada cuando se actualice la base de datos. (Más adelante, en la Sección 13.5, se describe el modo en que el sistema de bases de datos puede llevar a cabo de manera eficiente el mantenimiento de las vistas materializadas.)

Por desgracia, la mayoría de los sistemas de bases de datos no disponen de restricciones en vistas materializadas. Aunque la base de datos Oracle dispone de restricciones en vistas materializadas, por defecto realiza un mantenimiento de vistas cuando se accede a la vista, no cuando se actualiza la relación subyacente;[7] el resultado es que una violación de restricción se detecta después de realizarse la actualización, lo que hace inútil su detección.

Por tanto, en caso de que no se pueda obtener una descomposición en la FNBC que conserve las dependencias, suele resultar preferible optar por la FNBC, ya que la comprobación de dependencias funcionales distintas a las de restricciones de clave primaria es difícil en SQL.

7 Al menos para la versión de Oracle de 10g.

8.6. Descomposición mediante dependencias multivaloradas

No parece que algunos esquemas de relación, aunque se hallen en la FNBC, estén suficientemente normalizados, en el sentido de que siguen sufriendo el problema de la repetición de información. Considérese nuevamente el ejemplo de la universidad en el que un profesor pueda estar asociado a varios departamentos:

prof (*ID, nombre_dept, nombre, calle, ciudad*)

El lector avispado reconocerá este esquema como no correspondiente a la FNBC, debido a la dependencia funcional

ID → *nombre, calle, ciudad*

y a que *ID* no es clave de *prof*.

No obstante, supóngase que un profesor pueda tener varios domicilios (por ejemplo, una residencia de invierno y otra de verano). Entonces, ya no se desea que se cumpla la dependencia funcional «*ID* → *calle, ciudad*», aunque, por supuesto, se sigue deseando que se cumpla «*ID* → *nombre*» (es decir, la universidad no trata con profesores que operan con varios alias). A partir del algoritmo de descomposición en la FNBC se obtienen dos esquemas:

r_1 (*ID, nombre*)
r_2 (*ID, nombre_dept, calle, ciudad*)

Los dos se encuentran en la FNBC (recuérdese que cada profesor puede estar asociado a varios departamentos y un departamento tener varios profesores y, por tanto, no se cumple ni «*ID* → *nombre_dept*» ni «*nombre_dept* → *ID*»).

Pese a que r_2 se halla en la FNBC, hay redundancia. Se repite la dirección de cada residencia para cada profesor una vez por cada departamento que tenga asociado a ese profesor. Este problema se puede resolver descomponiendo más aún r_2 en:

r_{21} (*nombre_dept, ID*)
r_{22} (*ID, calle, ciudad*)

pero no hay ninguna restricción que nos lleve a hacerlo.

Para tratar este problema hay que definir una nueva modalidad de restricción, denominada *dependencia multivalorada*. Al igual que se hizo con las dependencias funcionales, se utilizarán las dependencias multivaloradas para definir una forma normal para los esquemas de relación. Esta forma normal, denominada **cuarta forma normal** (4FN), es más restrictiva que la FNBC. Se verá que cada esquema en la 4FN también se halla en la FNBC, pero hay esquemas en la FNBC que no se hallan en la 4FN.

8.6.1. Dependencias multivaloradas

Las dependencias funcionales impiden que ciertas tuplas estén en una relación dada. Si $A \rightarrow B$, entonces no puede haber dos tuplas con el mismo valor de A y diferentes valores de B. Las dependencias multivaloradas, por otro lado, no impiden la existencia de esas tuplas. Por el contrario, *exigen* que otras tuplas estén presentes en la relación de una forma determinada. Por este motivo, las dependencias funcionales a veces se denominan **dependencias que generan igualdades;** y las dependencias multivaloradas, **dependencias que generan tuplas.**

Sea r (R) un esquema de relación y sean $\alpha \subseteq R$ y $\beta \subseteq R$. La **dependencia multivalorada**

$$\alpha \rightarrow\rightarrow \beta$$

se cumple en R si, en cualquier relación legal $r(R)$ para todo par de tuplas t_1 y t_2 de r tales que $t_1[\alpha] = t_2[\alpha]$, existen unas tuplas t_3 y t_4 de r tales que

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Figura 8.13. Representación tabular de $\alpha \rightarrow\rightarrow \beta$.

Esta definición es menos complicada de lo que parece. La Figura 8.13 muestra una representación tabular de t_1, t_2, t_3 y t_4. De manera intuitiva, la dependencia multivalorada $\alpha \rightarrow\rightarrow \beta$ indica que la relación entre α y β es independiente de la relación entre α y $R - \beta$. Si todas las relaciones del esquema R satisfacen la dependencia multivalorada $\alpha \rightarrow\rightarrow \beta$, entonces $\alpha \rightarrow\rightarrow \beta$ es una dependencia multivalorada *trivial* del esquema R. Por tanto, $\alpha \rightarrow\rightarrow \beta$ es trivial si $\beta \subseteq \alpha$ o $\beta \cup \alpha = R$.

Para ilustrar la diferencia entre las dependencias funcionales y las multivaloradas, considérense de nuevo el esquema r_2 y la relación de ejemplo de ese esquema mostrada en la Figura 8.14. Hay que repetir el nombre de departamento una vez por cada dirección que tenga el profesor, y hay que repetir la dirección de cada departamento al que esté asociado un profesor. Esta repetición es innecesaria, ya que la relación entre cada profesor y su dirección es independiente de la relación entre ese profesor y el departamento. Si un profesor con identificador *ID* 99123 está asociado al departamento de Física, se desea que ese departamento esté asociado con todas las direcciones de ese profesor. Por tanto, la relación de la Figura 8.15 es ilegal. Para hacer que esa relación sea legal hay que añadir las tuplas (Física, 22222, Main, Manchester) y (Matemáticas, 22222, North, Rye) a la relación de la Figura 8.15.

Si se compara el ejemplo anterior con la definición de dependencia multivalorada, se ve que se desea que se cumpla la dependencia multivalorada:

ID →→ *calle, ciudad*

También se cumplirá la dependencia multivalorada *ID* →→ *nombre_dept*. Pronto se verá que son equivalentes.

Al igual que las dependencias funcionales, las dependencias multivaloradas se utilizan de dos maneras:

1. Para verificar las relaciones y determinar si son legales bajo un conjunto dado de dependencias funcionales y multivaloradas.

ID	*nombre_dept*	*calle*	*ciudad*
22222	Física	North	Rye
22222	Física	Main	Manchester
12121	Finanzas	Lake	Horseneck

Figura 8.14. Ejemplo de redundancia en una relación de un esquema en FNBC.

ID	*nombre_dept*	*calle*	*ciudad*
22222	Física	North	Rye
22222	Matemáticas	Main	Manchester

Figura 8.15. Relación r_2 ilegal.

2. Para especificar restricciones del conjunto de relaciones legales; de este modo, *solo* habrá que preocuparse de las relaciones que satisfagan un conjunto dado de dependencias funcionales y multivaloradas.

Téngase en cuenta que si una relación r no satisface una dependencia multivalorada dada, se puede crear una relación r' que *sí* la satisfaga añadiendo tuplas a r.

Supóngase que D denota un conjunto de dependencias funcionales y multivaloradas. El **cierre** D^+ de D es el conjunto de todas las dependencias funcionales y multivaloradas implicadas lógicamente por D. Al igual que se hizo con las dependencias funcionales, se puede calcular D^+ a partir de D, empleando las definiciones formales de dependencia funcional y multivalorada. Con este razonamiento se puede trabajar con dependencias multivaloradas muy sencillas. Afortunadamente, parece que las dependencias multivaloradas que se dan en la práctica son bastante sencillas. Para dependencias complejas es mejor razonar con conjuntos de dependencias mediante un sistema de reglas de inferencia.

A partir de la definición de dependencia multivalorada se pueden obtener las reglas siguientes para $\alpha, \beta \subseteq R$:

- Si $\alpha \rightarrow \beta$, entonces $\alpha \rightarrow\rightarrow \beta$. En otras palabras, toda dependencia funcional es también una dependencia multivalorada.
- Si $\alpha \rightarrow\rightarrow \beta$, entonces $\alpha \rightarrow\rightarrow R - \alpha - \beta$

En el Apéndice B.1.1 se describe un sistema de reglas de inferencia para dependencias multivaloradas.

8.6.2. Cuarta forma normal

Considérese nuevamente el ejemplo del esquema en la FNBC:

$$r_2\ (ID,\ nombre_dept,\ calle,\ ciudad)$$

en el que se cumple la dependencia multivalorada «$ID \rightarrow\rightarrow calle$, *ciudad*». Se vio en los primeros párrafos de la Sección 8.6 que, aunque este esquema se halla en la FNBC, el diseño no es el ideal, ya que hay que repetir la información sobre la dirección del profesor para cada departamento. Se verá que se puede utilizar esta dependencia multivalorada para mejorar el diseño de la base de datos, realizando la descomposición del esquema en la **cuarta forma normal.**

Un esquema de relación $r\ (R)$ está en la **cuarta forma normal** (4FN) con respecto a un conjunto D de dependencias funcionales y multivaloradas si para todas las dependencias multivaloradas de D^+ de la forma $\alpha \rightarrow\rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple, como mínimo, una de las condiciones siguientes:

- $\alpha \rightarrow\rightarrow \beta$ es una dependencia multivalorada trivial.
- α es superclave del esquema R.

El diseño de una base de datos está en la 4FN si cada componente del conjunto de esquemas de relación que constituye el diseño se halla en la 4FN.

Téngase en cuenta que la definición de la 4FN solo se diferencia de la definición de la FNBC en el empleo de las dependencias multivaloradas en lugar de las dependencias funcionales. Todos los esquemas en la 4FN están en la FNBC. Para verlo hay que darse cuenta de que si un esquema $r\ (R)$ no se halla en la FNBC, hay una dependencia funcional no trivial $\alpha \rightarrow \beta$ que se cumple en R en la que α no es superclave. Como $\alpha \rightarrow \beta$ implica $\alpha \rightarrow\rightarrow \beta$, R no puede estar en la 4FN.

Sea $r\ (R)$ un esquema de relación y sean $r_1\ (R_1), r_2\ (R_2) \ldots, r_n(R_n)$ una descomposición de $r\ (R)$. Para comprobar si cada esquema de relación r_i de la descomposición se halla en la 4FN hay que averiguar las dependencias multivaloradas que se cumplen en cada r_i. Recuérdese que, para un conjunto F de dependencias funcionales, la restricción F_i de F a R_i son todas las dependencias funcionales de F^+ que *solo* incluyen los atributos de R_i. Considérese ahora un conjunto D de dependencias funcionales y multivaloradas. La **restricción** de D a R_i es el conjunto D_i consistente en:

1. Todas las dependencias funcionales de D^+ que solo incluyen atributos de R_i
2. Todas las dependencias multivaloradas de la forma:

$$\alpha \rightarrow\rightarrow \beta \cap R_i$$

donde $\alpha \subseteq R_i$ y $\alpha \rightarrow\rightarrow \beta$ están en D^+.

8.6.3. Descomposición en la 4NF

La analogía entre la 4FN y la FNBC es aplicable al algoritmo para descomponer esquemas en la 4FN. La Figura 8.16 muestra el algoritmo de descomposición en la 4FN. Es idéntico al algoritmo de descomposición en la FNBC de la Figura 8.11, salvo porque emplea dependencias multivaloradas en lugar de funcionales y utiliza la restricción de D^+ a R_i.

Si se aplica el algoritmo de la Figura 8.16 a (*ID, nombre_dept, calle, ciudad*), se descubre que $ID \rightarrow\rightarrow nombre_dept$ es una dependencia multivalorada no trivial, y que *ID* no es superclave del esquema. De acuerdo con el algoritmo, se sustituye el esquema original por estos dos esquemas:

$$r_{21}\ (ID,\ nombre_dept)$$
$$r_{22}\ (ID,\ calle,\ ciudad)$$

Este par de esquemas, que se hallan en la 4FN, elimina la redundancia que se encontró anteriormente.

Como ocurría cuando se trataba solamente con las dependencias funcionales, resultan interesantes las descomposiciones que carecen de pérdidas y que conservan las dependencias. La siguiente circunstancia relativa a las dependencias multivaloradas y a la ausencia de pérdidas muestra que el algoritmo de la Figura 8.16 solo genera descomposiciones sin pérdidas:

```
resultado: = {R};
hecho: = falso;
calcular D+; dado el esquema Ri, Di denotará la restricción de D+ a Ri
while (not hecho) do
  if (hay un esquema Ri en resultado que no esté en 4FN respecto a Di)
    then begin
        sea α →→ β una dependencia multivalorada no trivial que se
        cumple en Ri tal que α → Ri no se halla en Di , y α ∩ β = ∅;
        resultado: = (resultado − Ri) ∪ (Ri − β ∪ (α, β);
      end
    else hecho: = cierto;
```

Figura 8.16. Algoritmo de descomposición en 4FN.

- Sea $r\ (R)$ un esquema de relación, y D un conjunto de dependencias funcionales y multivaloradas en R. Sean $r_1\ (R_1)$ y $r_2\ (R_2)$ una descomposición de R. Esta descomposición es sin pérdidas en R, si y solo si al menos una de las dependencias multivaloradas está en D^+:

$$R_1 \cap R_2 \rightarrow\rightarrow R_1$$
$$R_1 \cap R_2 \rightarrow\rightarrow R_2$$

Recuerde que ya se indicó en la Sección 8.4.4 que si $R_1 \cap R_2 \rightarrow R_1$ o $R_1 \cap R_2 \rightarrow R_2$, entonces $r_1\ (R_1)$ y $r_2\ (R_2)$ son descomposiciones sin pérdidas $r\ (R)$. Este hecho sobre dependencias multivaloradas es una afirmación más general sobre la característica sin pérdidas. Indica que para *todas* las descomposiciones sin pérdidas de $r\ (R)$ en dos esquemas $r_1\ (R_1)$ y $r_2\ (R_2)$, debe cumplirse una de las dos dependencias $R_1 \cap R_2 \rightarrow\rightarrow R_1$ o $R_1 \cap R_2 \rightarrow\rightarrow R_2$.

El tema de la conservación de la dependencia cuando se descompone un esquema de relación se vuelve más complicado en presencia de dependencias multivaloradas. En el Apéndice B.1.2 se trata este tema.

8.7. Más formas normales

La cuarta forma normal no es, de ningún modo, la forma normal «definitiva». Como ya se ha visto, las dependencias multivaloradas ayudan a comprender y a abordar algunas formas de repetición de la información que no pueden comprenderse en términos de las dependencias funcionales. Hay restricciones denominadas **depen-**

dencias de reunión que generalizan las dependencias multivaloradas y llevan a otra forma normal denominada **forma normal de reunión por proyección (FNRP)** (la FNRP se denomina en algunos libros **quinta forma normal**). Hay una clase de restricciones todavía más generales, que lleva a una forma normal denominada **forma normal de dominios y claves (FNDC).**

Un problema práctico del empleo de estas restricciones generalizadas es que no solo es difícil razonar con ellas, sino que tampoco hay un conjunto de reglas de inferencia seguras y completas para razonar sobre las restricciones. Por tanto, FNRP y FNDC se utilizan muy rara vez. El Apéndice B (B.2) ofrece más detalles sobre estas formas normales.

Destaca por su ausencia en este estudio de las formas normales la **segunda forma normal** (2FN). No se ha estudiado porque solo es de interés histórico. Simplemente se definirá, y se permitirá al lector experimentar con ella, en el Ejercicio práctico 8.17.

8.8. Proceso de diseño de la base de datos

Hasta ahora se han examinado aspectos detallados de las formas normales y de la normalización. En esta sección se estudiará el modo de encajar la normalización en el proceso global de diseño de las bases de datos.

Anteriormente, en este capítulo, a partir de la Sección 8.3, se ha dado por supuesto que se tenía un esquema de relación $r(R)$ y que se procedía a normalizarlo. Hay varios modos de obtener ese esquema $r(R)$:

1. $r(R)$ puede haberse generado al convertir un diagrama E-R en un conjunto de esquemas de relación.
2. $r(R)$ puede haber sido una sola relación que contenía *todos* los atributos que resultaban de interés. El proceso de normalización divide a $r(R)$ en relaciones más pequeñas.
3. $r(R)$ puede haber sido el resultado de algún diseño ad hoc de las relaciones, que hay que comprobar para asegurarse de que satisface la forma normal deseada.

En el resto de esta sección se examinarán las implicaciones de estos enfoques. También se examinarán algunos aspectos prácticos del diseño de las bases de datos, incluida la desnormalización para el rendimiento y ejemplos de mal diseño que no son detectados por la normalización.

8.8.1. El modelo E-R y la normalización

Cuando se definen con cuidado los diagramas E-R, identificando correctamente todas las entidades, los esquemas de relación generados a partir de ellos no deben necesitar mucha más normalización. No obstante, puede haber dependencias funcionales entre los atributos de alguna entidad. Por ejemplo, supóngase que la entidad *profesor* tiene los atributos *nombre_dept* y *dirección_departamento,* y que hay una dependencia funcional *nombre_dept* → *dirección_departamento*. Habrá que normalizar la relación generada a partir de *profesor*.

La mayor parte de los ejemplos de este tipo de dependencias surge de un mal diseño del diagrama E-R. En el ejemplo anterior, si se hubiera diseñado correctamente el diagrama E-R, se habría creado una entidad *departamento* con el atributo *dirección_departamento* y una relación entre *profesor* y *departamento*. De manera parecida, puede que una relación que implique a más de dos entidades no se halle en la forma normal deseable. Como la mayor parte de las relaciones son binarias, estos casos resultan relativamente raros (de hecho, algunas variantes de los diagramas E-R hacen realmente difícil o imposible especificar relaciones no binarias).

Las dependencias funcionales pueden ayudar a detectar un mal diseño E-R. Si las relaciones generadas no se hallan en la forma normal deseada, el problema puede solucionarse en el diagrama E-R. Es decir, la normalización puede llevarse a cabo formalmente como parte del modelado de los datos. De manera alternativa, la normalización puede dejarse a la intuición del diseñador durante el modelado E-R, y puede hacerse formalmente sobre las relaciones generadas a partir del modelo E-R.

El lector atento se habrá dado cuenta de que para que se pudiera ilustrar la necesidad de las dependencias multivaloradas y de la cuarta forma normal hubo que comenzar con esquemas que no se obtuvieron a partir del diseño E-R. En realidad, el proceso de creación de diseños E-R tiende a generar diseños 4FN. Si se cumple alguna dependencia multivalorada y no la implica la dependencia funcional correspondiente, suele proceder de alguna de las fuentes siguientes:

- Una relación de varios a varios.
- Un atributo multivalorado de un conjunto de entidades.

En las relaciones de varios a varios cada conjunto de entidades relacionado tiene su propio esquema y hay un esquema adicional para el conjunto de relaciones. Para los atributos multivalorados se crea un esquema diferente que consta de ese atributo y de la clave primaria del conjunto de entidades (como en el caso del atributo *número_teléfono* del conjunto de entidades *profesor*).

El enfoque de las relaciones universales para el diseño de bases de datos relacionales parte de la suposición de que solo hay un esquema de relación que contenga todos los atributos de interés. Este esquema único define la manera en que los usuarios y las aplicaciones interactúan con la base de datos.

8.8.2. Denominación de los atributos y las relaciones

Una característica deseable del diseño de bases de datos es la **asunción de un rol único,** lo que significa que cada nombre de atributo tiene un significado único en toda la base de datos. Esto evita que se utilice el mismo atributo para indicar cosas diferentes en esquemas diferentes. Por ejemplo, puede que, de otra manera, se considerara el uso del atributo *número* para el número de teléfono en el esquema *profesor* y para el número de aula en el esquema *aula*. La reunión de una relación del esquema *profesor* con otra de *aula* carecería de significado. Aunque los usuarios y los desarrolladores de aplicaciones pueden trabajar con esmero para garantizar el uso apropiado de *número* en cada circunstancia, tener nombres de atributo diferentes para el número de teléfono y para el de aula sirve para reducir los errores de los usuarios.

Es buena idea hacer que los nombres de los atributos incompatibles sean diferentes, pero si los atributos de relaciones diferentes tienen el mismo significado, puede ser conveniente emplear el mismo nombre de atributo. Por ejemplo, se ha utilizado el nombre de atributo «*nombre*» para los conjuntos de entidades *profesor* y *estudiante*. Si no fuese así, es decir, si hubiésemos convenido una denominación diferente para los nombres de profesor y estudiante, entonces cuando deseáramos generalizar esos conjuntos de entidades mediante la creación del conjunto de entidades *persona*, habría que renombrar el atributo. Por tanto, aunque no se tenga actualmente una generalización de *estudiante* y de *profesor*, si se prevé esa posibilidad es mejor emplear el mismo nombre en los dos conjuntos de relaciones (y en sus relaciones).

Aunque, técnicamente, el orden de los nombres de los atributos en los esquemas no tiene ninguna importancia, es costumbre relacionar en primer lugar los atributos de la clave primaria. Esto facilita la lectura de los resultados predeterminados (como los generados por **select** *).

En los esquemas de bases de datos de gran tamaño, los conjuntos de relaciones (y los esquemas derivados) se suelen denominar mediante la concatenación de los nombres de los conjuntos de entidades a los que hacen referencia, quizás con guiones o caracteres de subrayado intercalados. En este libro se han utilizado algunos nombres de este tipo, por ejemplo, *prof_secc* y *estudiante_secc*. Se han utilizado los nombres *enseña* o *matricula* en lugar de otros concatenados de mayor longitud y resulta aceptable, ya que no es difícil recordar las entidades asociadas a unas pocas relaciones. Sin embargo, no siempre se pueden crear nombres de relaciones mediante la mera concatenación; por ejemplo, la relación jefe o trabaja-para entre empleados no tendría mucho sentido si se llamara *empleado_empleado*. De manera parecida, si hay varios conjuntos de relaciones posibles entre un par de conjuntos de entidades, los nombres de las relaciones deben incluir componentes adicionales para identificar cada relación.

Las diferentes organizaciones tienen costumbres diferentes para la denominación de las entidades. Por ejemplo, a un conjunto de entidades de estudiantes se le puede denominar *estudiante* o *estudiantes*. En los diseños de las bases de datos de este libro se ha decidido utilizar la forma en singular. Es aceptable tanto el empleo del singular como el del plural, siempre y cuando la convención se utilice de manera consistente en todas las entidades.

A medida que los esquemas aumentan de tamaño, con un número creciente de relaciones, el empleo de una denominación consistente de los atributos, de las relaciones y de las entidades facilita mucho la vida de los diseñadores de bases de datos y de los programadores de aplicaciones.

8.8.3. Desnormalización para el rendimiento

A veces, los diseñadores de bases de datos escogen un esquema que tiene información redundante; es decir, que no está normalizado. Utilizan la redundancia para mejorar el rendimiento de aplicaciones concretas. La penalización sufrida por no emplear un esquema normalizado es el trabajo adicional (en términos de tiempos de codificación y de ejecución) de mantener consistentes los datos redundantes.

Por ejemplo, supóngase que hay que mostrar todos los prerrequisitos de las asignaturas junto con la información de la misma, cada vez que se accede a una asignatura. En el esquema normalizado esto exige una reunión de *asignatura* con *prerreq*.

Una alternativa al cálculo de la reunión sobre la marcha es almacenar una relación que contenga todos los atributos de *asignatura* y de *prerreq*. Esto hace más rápida la visualización de la información «completa» de la asignatura. Sin embargo, la información de la asignatura se repite para cada uno de los prerrequisitos de la misma, y la aplicación debe actualizar todas las copias cada vez que se añada o elimine una asignatura. El proceso de tomar un esquema normalizado y hacer que no esté normalizado se denomina **desnormalización,** y los diseñadores lo utilizan para ajustar el rendimiento de los sistemas para que den soporte a las operaciones críticas en el tiempo.

Una opción mejor, soportada hoy en día por muchos sistemas de bases de datos, es emplear el esquema normalizado y, además, almacenar la reunión de *asignatura* y *prerreq* en forma de vista materializada. (Recuérdese que las vistas materializadas son vistas cuyo resultado se almacena en la base de datos y se actualiza cuando se actualizan las relaciones utilizadas en ellas). Al igual que la desnormalización, el empleo de las vistas materializadas supone sobrecargas de espacio y de tiempo; sin embargo, presenta la ventaja de que conservar actualizadas las vistas es labor del sistema de bases de datos, no del programador de la aplicación.

8.8.4. Otros problemas de diseño

Hay algunos aspectos del diseño de bases de datos que la normalización no aborda y, por tanto, pueden llevar a un mal diseño de la base de datos. Los datos relativos al tiempo o a intervalos temporales presentan varios de esos problemas. A continuación se ofrecen algunos ejemplos; evidentemente, conviene evitar esos diseños.

Considérese una base de datos de una universidad, en la que se desea almacenar el número total de profesores de cada departamento a lo largo de varios años. Se puede utilizar la relación *total_prof* (*nombre_dept*, *año*, *total*) para almacenar la información deseada. La única dependencia funcional de esta relación es *nombre_dept*, *año* → *total*, y se halla en la FNBC.

Un diseño alternativo es el empleo de varias relaciones, cada una de las cuales almacena la información del total para cada año. Supóngase que los años que nos interesan son 2007, 2008 y 2009; se tendrán, entonces, relaciones de la forma *total_prof_2007*, *total_prof_2008* y *total_prof_2009*, todas las cuales se hallan en el esquema (*nombre_dept*, *total*). En este caso, la única dependencia funcional de cada relación será *nombre_dept* → *total*, por lo que esas relaciones también se hallan en la FNBC.

No obstante, este diseño alternativo es, claramente, una mala idea; habría que crear una relación nueva cada año, y también habría que escribir consultas nuevas todos los años, para tener en cuenta cada nueva relación. Las consultas también serían más complicadas, ya que probablemente tendrían que hacer referencia a muchas relaciones.

Otra manera de representar esos mismos datos es tener una sola relación *dept_año* (*nombre_dept*, *total_prof_2007*, *total_prof_2008*, *total_prof_2009*). En este caso, las únicas dependencias funcionales van de *nombre_dept* hacia los demás atributos y, una vez más, la relación se halla en la FNBC. Este diseño también es una mala idea, ya que plantea problemas parecidos a los del diseño anterior; es decir, habría que modificar el esquema de la relación y escribir consultas nuevas cada año. Las consultas también serían más complicadas, ya que puede que tuvieran que hacer referencia a muchos atributos.

Las representaciones como las de la relación *dept_año*, con una columna para cada valor de cada atributo, se denominan de **referencias cruzadas;** se emplean mucho en las hojas de cálculo, en los informes y en las herramientas de análisis de datos. Aunque esas representaciones resultan útiles para mostrárselas a los usuarios, por las razones que se acaban de dar, no resultan deseables en el diseño de bases de datos. Se han propuesto extensiones del SQL para pasar los datos de la representación relacional normal a la de referencias cruzadas, para su visualización, como se ha tratado en la Sección 5.6.2.

8.9. Modelado de datos temporales

Supóngase que en la universidad se conservan datos que no solo muestran la dirección de cada profesor, sino también todas las direcciones anteriores de las que la universidad tenga noticia. Se pueden formular consultas como «Averiguar todos los profesores que vivían en Princeton en 1981». En ese caso, puede que se tengan varias direcciones para cada profesor. Cada dirección tiene asociadas una fecha de comienzo y otra de finalización, que indican el periodo en que el profesor residió en esa dirección. Se puede utilizar un valor especial para la fecha de finalización, por ejemplo, nulo, o un valor que se halle claramente en el futuro, como 31/12/9999, para indicar que el profesor sigue residiendo en esa dirección.

En general, los **datos temporales** son datos que tienen asociado un intervalo de tiempo durante el cual son **válidos.**[8] Se utiliza el término **instantánea** de los datos para indicar su valor en un

8 Hay otros modelos de datos temporales que distinguen entre **periodo de validez** y **momento de la transacción;** el último registra el momento en que se registró un hecho en la base de datos. Para simplificar se prescinde de estos detalles.

momento determinado. Por tanto, una instantánea de los datos de *asignatura* muestra el valor de todos los atributos, como su nombre y departamento, en un momento concreto.

El modelado de datos temporales es una cuestión interesante por varios motivos. Por ejemplo, supóngase que se tiene una entidad *profesor* con la que se desea asociar una dirección que varía con el tiempo. Para añadir información temporal a una dirección hay que crear un atributo multivalorado, cada uno de cuyos valores es un valor compuesto que contiene una dirección y un intervalo de tiempo. Además de los valores de los atributos que varían con el tiempo, puede que las propias entidades tengan un periodo de validez asociado. Por ejemplo, la entidad estudiante puede tener un periodo de validez desde la fecha de primera matrícula en la universidad hasta que se gradúa (o abandona la universidad). Las relaciones también pueden tener asociados periodos de validez. Por ejemplo, la relación *prerreq* puede registrar el momento en que una asignatura se convierte en prerrequisito de otra. Por tanto, habría que añadir intervalos de validez a los valores de los atributos, a las entidades y a las relaciones. La adición de esos detalles a los diagramas E-R hace que sean muy difíciles de crear y de comprender. Ha habido varias propuestas para extender la notación E-R para que especifique de manera sencilla que un atributo o una relación varía con el tiempo, pero no hay ninguna norma aceptada al respecto.

Cuando se realiza un seguimiento del valor de los datos a lo largo del tiempo, puede que dejen de cumplirse dependencias funcionales que se suponían cumplidas, por ejemplo:

$$ID \rightarrow calle, ciudad$$

puede que no se cumpla. En su lugar, se cumpliría la restricción siguiente (expresada en castellano): «Un profesor *ID* solo tiene un valor de *calle* y de *ciudad* para cada momento *t* dado».

Las dependencias funcionales que se cumplen en un momento concreto se denominan dependencias funcionales temporales. Formalmente, la **dependencia funcional temporal** $X \xrightarrow{\tau} Y$ se cumple en un esquema de relación $r(R)$ si, para todos los ejemplares legales de $r(R)$, todas las instantáneas de r satisfacen la dependencia funcional $X \rightarrow Y$.

Se puede extender la teoría del diseño de bases de datos relacionales para que tenga en cuenta las dependencias funcionales temporales. Sin embargo, el razonamiento con las dependencias funcionales normales ya resulta bastante difícil, y pocos diseñadores están preparados para trabajar con las dependencias funcionales temporales.

En la práctica, los diseñadores de bases de datos recurren a enfoques más sencillos para diseñar las bases de datos temporales. Un enfoque empleado con frecuencia es diseñar toda la base de datos (incluidos los diseños E-R y relacional) ignorando las modificaciones temporales (o, lo que es lo mismo, tomando solo una instantánea en consideración). Tras esto, el diseñador estudia las diferentes relaciones y decide las que necesitan un seguimiento de su variación temporal.

El paso siguiente es añadir información sobre los periodos de validez a cada una de esas relaciones, añadiendo como atributos el momento de inicio y el de finalización. Por ejemplo, supóngase que se tiene la relación *asignatura*. El nombre de la asignatura puede cambiar con el tiempo, que se puede tratar añadiendo un periodo de tiempo, el esquema resultante sería

asignatura (*asignatura_id*, *nombre_asig*,
nombre_dept, *inicio*, *fin*)

Un ejemplar de esta relación puede tener dos registros (CS101, «Introducción a la programación», 01/01/1985, 31/12/2000) y (CS101, «Introdución a C», 01/01/2001, 31/12/9999). Si se actualiza la relación para cambiar la denominación de la asignatura a «Introducción a Java», se actualizaría la fecha «31/12/9999» a la correspondiente al momento hasta el que fue válido el valor anterior («Introducción a C»), y se añadiría una tupla nueva que contendría la nueva denominación («Introducción a Java»), con la fecha de inicio correspondiente.

Si otra relación tuviera una clave externa que hiciera referencia a una relación temporal, el diseñador de la base de datos tendría que decidir si la referencia se hace a la versión actual de los datos o a los datos de un momento concreto. Por ejemplo, se puede extender la relación *departamento* para registrar los cambios en el edificio o el presupuesto de un departamento en el tiempo, pero una referencia de la relación *profesor* o *estudiante* puede no tener nada que ver con la historia del edificio o el presupuesto, pero se puede referir de forma implícita al registro temporal actual para el correspondiente *nombre_dept*. Por otro lado, los registros del expediente de cada estudiante deben hacer referencia a la denominación de la asignatura en el momento en que la cursó ese estudiante. En este último caso, la relación que hace la referencia también debe almacenar la información temporal, para identificar cada registro concreto de la relación *asignatura*. En nuestro ejemplo, el *año* y *semestre* en que la asignatura se impartió pueden aludir a un valor de hora/fecha como desde medianoche de la fecha de inicio del semestre; el valor resultante de hora/fecha se utiliza para identificar un registro en particular a partir de la versión temporal de la relación *asignatura*, desde la que se obtiene el *nombre*.

La clave primaria original de una relación temporal ya no identificaría de manera unívoca a cada tupla. Para solucionar este problema se pueden añadir a la clave primaria los atributos de fecha de inicio y de finalización. Sin embargo, persisten algunos problemas:

- Es posible almacenar datos con intervalos que se solapen, pero la restricción de clave primaria no puede detectarlo. Si el sistema soporta un tipo nativo *periodo de validez*, puede detectar y evitar esos intervalos temporales que se solapan.
- Para especificar una clave externa que haga referencia a una relación así, las tuplas que hacen la referencia tienen que incluir los atributos de momento inicial y final como parte de su clave externa, y los valores deberán coincidir con los de la tupla a la que hacen referencia. Además, si la tupla a la que hacen referencia se actualiza (y el momento final que se hallaba en el futuro se actualiza), esa actualización debe propagarse a todas las tuplas que hacen referencia a ella.

  Si el sistema soporta los datos temporales de alguna manera mejor, se puede permitir que la tupla que hace la referencia especifique un momento, en lugar de un rango temporal, y confiar en que el sistema garantice que hay una tupla en la relación a la que hace referencia cuyo periodo de validez contenga ese momento. Por ejemplo, un registro de un expediente puede especificar una *asignatura_id* y un momento (como, la fecha de comienzo de un trimestre), lo que basta para identificar el registro correcto de la relación *asignatura*.

Como caso especial frecuente, si todas las referencias a los datos temporales se hacen exclusivamente a los datos actuales, una solución más sencilla es no añadir información temporal a la relación y, en su lugar, crear la relación *historial* correspondiente que contenga esa información temporal, para los valores del pasado. Por ejemplo, en la base de datos bancaria, se puede utilizar el diseño que se ha creado, ignorando las modificaciones temporales, para almacenar únicamente la información actual. Toda la información histórica se traslada a las relaciones históricas. Por tanto, la relación *profesor* solo puede almacenar la dirección actual, mientras que la relación *historial_profesor* puede contener todos los atributos de *profesor*, con los atributos adicionales *momento_inicial* y *momento_final*.

Aunque no se ha ofrecido ningún método formal para tratar los datos temporales, los problemas estudiados y los ejemplos ofrecidos deben ayudar al lector a diseñar bases de datos que registren datos temporales. Más adelante, en la Sección 25.2, se tratan otros aspectos del manejo de datos temporales, incluidas las consultas temporales.

8.10. Resumen

- Se han mostrado algunas dificultades del diseño de bases de datos y el modo de diseñar de manera sistemática esquemas de bases de datos que eviten esas dificultades. Entre esas dificultades están la información repetida y la imposibilidad de representar cierta información.
- Se ha mostrado el desarrollo del diseño de bases de datos relacionales a partir de los diseños E-R, cuándo los esquemas se pueden combinar con seguridad y cuándo se deben descomponer. Todas las descomposiciones válidas deben ser sin pérdidas.
- Se han descrito las suposiciones de dominios atómicos y de primera forma normal.
- Se ha introducido el concepto de las dependencias funcionales y se ha utilizado para presentar dos formas normales, la forma normal de Boyce–Codd (FNBC) y la tercera forma normal (3FN).
- Si la descomposición conserva las dependencias, dada una actualización de la base de datos, todas las dependencias funcionales pueden verificarse a partir de las diferentes relaciones, sin necesidad de calcular la reunión de las relaciones de la descomposición.
- Se ha mostrado la manera de razonar con las dependencias funcionales. Se ha puesto un énfasis especial en señalar las dependencias que están implicadas lógicamente por conjuntos de dependencias. También se ha definido el concepto de recubrimiento canónico, que es un conjunto mínimo de dependencias funcionales equivalente para un conjunto dado de dependencias funcionales.
- Se ha descrito un algoritmo para la descomposición de las relaciones en la FNBC. Hay relaciones para las cuales no hay ninguna descomposición en la FNBC que conserve las dependencias.
- Se han utilizado los recubrimientos canónicos para descomponer las relaciones en la 3FN, que es una pequeña relajación de las condiciones de la FNBC. Las relaciones en la 3FN pueden tener alguna redundancia, pero siempre hay una descomposición en la 3FN que conserva las dependencias.
- Se ha presentado el concepto de dependencias multivaloradas, que especifican las restricciones que no pueden especificarse únicamente con las dependencias funcionales. Se ha definido la cuarta forma normal (4FN) con las dependencias multivaloradas. El Apéndice B.1.1 ofrece detalles del razonamiento sobre las dependencias multivaloradas.
- Otras formas normales, como la FNRP y la FNDC, eliminan formas más sutiles de redundancia. Sin embargo, es difícil trabajar con ellas y se emplean rara vez. El Apéndice B.2 ofrece detalles sobre estas formas normales.
- Al revisar los temas de este capítulo hay que tener en cuenta que el motivo de que se hayan podido definir enfoques rigurosos del diseño de bases de datos relacionales es que el modelo relacional de datos descansa sobre una base matemática sólida. Esa es una de las principales ventajas del modelo relacional en comparación con los otros modelos de datos que se han estudiado.

Términos de repaso

- Modelo E-R y normalización.
- Descomposición.
- Dependencias funcionales.
- Descomposición sin pérdidas.
- Dominios atómicos.
- Primera forma normal (1FN).
- Relaciones legales.
- Superclave.
- R satisface F.
- F se cumple en R.
- Forma normal de Boyce-Codd (FNBC).
- Conservación de las dependencias.
- Tercera forma normal (3FN).
- Dependencias funcionales triviales .
- Cierre de un conjunto de dependencias funcionales.
- Axiomas de Armstrong.
- Cierre de conjuntos de atributos.
- Restricción de F a R_i.
- Recubrimiento canónico.
- Atributos raros.
- Algoritmo de descomposición en la FNBC.
- Algoritmo de descomposición en la 3FN.
- Dependencias multivaloradas.
- Cuarta forma normal (4FN).
- Restricción de las dependencias multivaloradas.
- Forma normal de reunión por proyección (FNRP).
- Forma normal de dominios y claves (FNDC).
- Relación universal.
- Suposición de un rol único.
- Desnormalización.

Ejercicios prácticos

8.1. Supóngase que se descompone el esquema $r(A, B, C, D, E)$ en:

$$r_1(A, B, C)$$
$$r_2(A, D, E)$$

Demuéstrese que esta descomposición es una descomposición sin pérdidas si se cumple el siguiente conjunto F de dependencias funcionales:

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

8.2. Indíquense todas las dependencias funcionales que satisface la relación de la Figura 8.17.

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

Figura 8.17. Relación para el Ejercicio práctico 8.2.

8.3. Explíquese el modo en que se pueden utilizar las dependencias funcionales para indicar que:

- Existe un conjunto de relaciones de uno a uno entre los conjuntos de entidades *estudiante* y *profesor*.
- Existe un conjunto de relaciones de varios a uno entre los conjuntos de entidades *estudiante* y *profesor*.

8.4. Empléense los axiomas de Armstrong para probar la corrección de la regla de la unión. *Sugerencia:* utilícese la regla de la aumentatividad para probar que, si $\alpha \rightarrow \beta$, entonces $\alpha \rightarrow \alpha\beta$. Aplíquese nuevamente la regla de la aumentatividad, utilizando $\alpha \rightarrow \gamma$, y aplíquese luego la regla de la transitividad.

8.5. Empléense los axiomas de Armstrong para probar la corrección de la regla de la pseudotransitividad.

8.6. Calcúlese el cierre del siguiente conjunto F de relaciones funcionales para el esquema de relación $r(A, B, C, D, E)$.

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

Indíquense las claves candidatas de R.

8.7. Utilizando las dependencias funcionales del Ejercicio práctico 8.6, calcúlese el recubrimiento canónico F_c.

8.8. Considérese el algoritmo de la Figura 8.18 para calcular α^+. Demuéstrese que este algoritmo resulta más eficiente que el presentado en la Figura 8.8 (Sección 8.4.2) y que calcula α^+ de manera correcta.

8.9. Dado el esquema de base de datos $R(a, b, c)$ y una relación r del esquema R, escríbase una consulta SQL para comprobar si la dependencia funcional $b \rightarrow c$ se cumple en la relación r. Escríbase también un aserto de SQL que haga que se cumpla la dependencia funcional. Supóngase que no hay ningún valor nulo. (Aunque forma parte del estándar SQL, no existe ninguna implementación de base de datos que soporte estas aserciones).

8.10. Lo tratado sobre la descomposición de reunión sin pérdidas supone implícitamente que los atributos de la parte izquierda de una dependencia funcional no pueden tomar el valor nulo. ¿Qué fallaría en la descomposición si no se cumple esta propiedad?

8.11. En el algoritmo de descomposición en FNBC, suponga que se utiliza la dependencia funcional $\alpha \rightarrow \beta$ para descomponer un esquema de relación $r(\alpha, \beta, \gamma)$ en $r_1(\alpha, \beta)$ y $r_2(\alpha, \gamma)$.

a. ¿Cuáles son las restricciones de claves primaria y externa que se deberían cumplir en las relaciones descompuestas?

b. Indique un ejemplo de inconsistencia, que se puede producir a causa de una actualización errónea, si la restricción de clave externa no se fuerza en las relaciones descompuestas anteriores.

c. Cuando una relación se descompone en 3FN utilizando el algoritmo de la Sección 8.5.2, ¿qué dependencias de claves primaria y externa se debería esperar que se cumpliesen en el esquema descompuesto?

8.12. Sea $R_1, R_2 \ldots, R_n$ una descomposición del esquema U. Sea $u(U)$ una relación, y sea $r_i = \Pi R_I(u)$. Demuéstrese que:

$$u \subseteq r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$$

8.13. Demuéstrese que la descomposición del Ejercicio práctico 8.1 no es una descomposición que conserve las dependencias.

```
resultado: = ∅;
/* fdcount es una matriz (array) cuyo elemento i-ésimo contiene
el número de atributos del lado izquierdo de la i-ésima
DF que todavía no se sabe que estén en α+ */
for i: = 1 para |F| do
  begin
    Supóngase que β → γ denota la i-ésima DF;
    fdcount [i]: = |β|;
  end
/* appears es una matriz con una entrada por cada atributo. La entrada
del atributo A es una lista de enteros. Cada entero i de la lista indica que
A aparece en el lado izquierdo de la i-ésima DF */
for each atributo A do
  begin
    appears [A]: = NI L;
    for i: = 1 para |F| do
      begin
        Supóngase que β → γ denota la i-ésima DF;
        if A ∈ β then añadir i a appears [A];
      end
  end
addin (α);
return (resultado);
procedure addin (α);
for each atributo A en α do
  begin
    if A ∈ resultado then
      begin
        resultado: = resultado ∪ {A};
        for each elemento i de appears [A] do
          begin
            fdcount [i]: = fdcount [i] – 1;
            if fdcount [i]: = 0 then
              begin
                Supóngase que β → γ denota la i-ésima DF;
                addin (γ);
              end
          end
      end
  end
```

Figura 8.18. Algoritmo para el cálculo de α^+.

8.14. Demuéstrese que es posible asegurar que una descomposición que conserve las dependencias en la 3FN sea una descomposición sin pérdidas garantizando que, como mínimo, un esquema contenga una clave candidata para el esquema que se está descomponiendo. *Sugerencia:* demuéstrese que la reunión de todas las proyecciones en los esquemas de la descomposición no puede tener más tuplas que la relación original.

8.15. Indique un ejemplo de esquema de relación R' y un conjunto F' de dependencias funcionales (DF) tales que haya, al menos, tres descomposiciones sin pérdidas distintas de R' en FNBC.

8.16. Sea atributo **primo** el que aparece, como mínimo, en una clave candidata. Sean α y β conjuntos de atributos tales que se cumple $\alpha \rightarrow \beta$, pero no se cumple $\beta \rightarrow \alpha$. Sea A un atributo que no esté en α ni en β y para el que se cumple que $\beta \rightarrow \alpha$. Se dice que A es **dependiente de manera transitiva** de α. Se puede reformular la definición de la 3FN de la manera siguiente: *el esquema de relación R está en la 3FN con respecto al conjunto F de dependencias funcionales si no existen atributos no primos A en R para los cuales A sea dependiente de manera transitiva de una clave de R. Demuéstrese que esta nueva definición es equivalente a la original.*

8.17. La dependencia funcional $\alpha \rightarrow \beta$ se denomina **dependencia parcial** si hay un subconjunto propio γ de α tal que $\gamma \rightarrow \beta$. Se dice que β es *parcialmente dependiente* de α. El esquema de relación R está en la **segunda forma normal** (2FN) si cada atributo A de R cumple uno de los criterios siguientes:

- Aparece en una clave candidata
- No es parcialmente dependiente de una clave candidata.

Demuéstrese que cada esquema en la 3FN se halla en la 2FN. *Sugerencia:* demuéstrese que todas las dependencias parciales son dependencias transitivas.

8.18. Dé un ejemplo de esquema de relación R y un conjunto de dependencias tales que R se halle en la FNBC, pero no en la 4FN.

Ejercicios

8.19. Indique una descomposición de reunión sin pérdidas en FNBC del esquema R del Ejercicio práctico 8.1.

8.20. Indique una descomposición de reunión sin perdidas, que conserva las dependencias en 3FN del esquema R del Ejercicio práctico 8.1.

8.21. Normalice el siguiente esquema, con las restricciones indicadas, a 4FN.

libros (*acceso_num, isbn, título, autor, editorial*)
usuarios (*usuario_id, nombre, dept_id, dept_nombre*)
acceso_num → *isbn*
isbn → *título*
isbn → *editorial*
isbn →→ *autor*
userid → *nombre*
userid → *dept_id*
deptid → *dept_nombre*

8.22. Explíquese lo que se quiere decir con *repetición de la información* e *imposibilidad de representación de la información*. Explíquese el motivo por el que estas propiedades pueden indicar un mal diseño de las bases de datos relacionales.

8.23. Indíquese el motivo de que ciertas dependencias funcionales se denominen dependencias funcionales *triviales*.

8.24. Utilícese la definición de dependencia funcional para argumentar que cada uno de los axiomas de Armstrong (reflexividad, aumentatividad y transitividad) es correcto.

8.25. Considérese la siguiente regla propuesta para las dependencias funcionales: si $\alpha \rightarrow \beta$ y $\gamma \rightarrow \beta$, entonces $\alpha \rightarrow \gamma$. Pruébese que esta regla no es correcta mostrando una relación r que satisfaga $\alpha \rightarrow \beta$ y $\gamma \rightarrow \beta$, pero no $\alpha \rightarrow \gamma$.

8.26. Utilícense los axiomas de Armstrong para probar la corrección de la regla de la descomposición.

8.27. Utilizando las dependencias funcionales del Ejercicio práctico 8.6, calcúlese B^+.

8.28. Demuéstrese que la siguiente descomposición del esquema R del Ejercicio práctico 8.1 no es una descomposición sin pérdidas:

$$(A, B, C)$$
$$(C, D, E)$$

Sugerencia: indique un ejemplo de una relación r del esquema R tal que:

$$\Pi_{A, B, C}(r) \bowtie \Pi_{C, D, E}(r) \neq r$$

8.29. Considérese el siguiente conjunto F de dependencias funcionales en el esquema de relación $r(A, B, C, D, E, F)$:

$$A \rightarrow BCD$$
$$BC \rightarrow DE$$
$$B \rightarrow D$$
$$D \rightarrow A$$

a. Calcule B^+.

b. Demuestre (usando los axiomas de Armstrong) que AF es una superclave.

c. Calcule un recubrimiento canónico para el conjunto de dependencias funcionales F anterior; indique los pasos de la derivación con las correspondientes explicaciones.

d. Indique una descomposición en 3FN de r basada en el recubrimiento canónico.

e. Indique una descomposición en FNBC de r utilizando el conjunto original de dependencias funcionales.

f. ¿Se podría obtener la misma descomposición en FNBC de r anterior, utilizando el recubrimiento canónico?

8.30. Indíquense los tres objetivos de diseño de las bases de datos relacionales y explíquese el motivo de que cada uno de ellos sea deseable.

8.31. En el diseño de una base de datos relacional, ¿por qué se podría llegar a elegir un diseño que no estuviese en FNBC?

8.32. Dados los tres objetivos de diseño de las bases de datos relacionales, ¿hay alguna razón para diseñar un esquema en la 2FN, pero que no esté en ninguna forma normal superior? (Consulte el Ejercicio práctico 8.17 para ver la definición de 2FN).

8.33. Dado el esquema relacional $r(A, B, C, D)$, ¿implica lógicamente $A \rightarrow\rightarrow BC$ a $A \rightarrow\rightarrow B$ y a $A \rightarrow\rightarrow C$? En caso positivo, pruébese; en caso contrario, ofrézcase un contraejemplo.

8.34. Explíquese el motivo de que la 4FN sea una forma normal más deseable que la FNBC.

Notas bibliográficas

El primer estudio de la teoría del diseño de bases de datos relacionales apareció en un artículo pionero de Codd [1970]. En ese artículo, Codd introducía también las dependencias funcionales y la primera, la segunda y la tercera formas normales.

Los axiomas de Armstrong se introdujeron en Armstrong [1974]. A finales de los años setenta se produjo un desarrollo significativo de la teoría de las bases de datos relacionales. Esos resultados se recogen en varios textos sobre la teoría de las bases de datos, como Maier [1983], Atzeni y Antonellis [1993] y Abiteboul et ál. [1995].

La FNBC se introdujo en Codd [1972]. Biskup et ál. [1979] dan el algoritmo que se ha utilizado para encontrar descomposiciones en la 3FN que conserven las dependencias. Los resultados fundamentales de la propiedad de la descomposición sin pérdidas aparecen en Aho et ál. [1979a].

Beeri et ál. [1977] dan un conjunto de axiomas para las dependencias multivaloradas y prueban que los axiomas de los autores son correctos y completos. Los conceptos de 4FN, de FNRP y de FNDC son de Fagin [1977], Fagin [1979] y Fagin [1981], respectivamente. Véanse las notas bibliográficas del Apéndice B para tener más referencias sobre la literatura relativa a la normalización.

Jensen et ál. [1994] presentan un glosario de conceptos relacionados con las bases de datos temporales. Gregersen y Jensen [1999] presentan un resumen de las extensiones del modelo E-R para el tratamiento de datos temporales. Tansel et ál. [1993] tratan la teoría, el diseño y la implementación de las bases de datos temporales. Jensen et ál. [1996] describen las extensiones de la teoría de la dependencia a los datos temporales.

Diseño y desarrollo de aplicaciones

09

Casi todo el uso de las bases de datos se produce desde los programas de aplicación. A su vez, casi toda la interacción de los usuarios con las bases de datos es indirecta, mediante los programas de aplicación. No resulta sorprendente, por tanto, que los sistemas de bases de datos lleven mucho tiempo soportando herramientas como los generadores de formularios y de interfaces gráficas de usuario, que ayudan a lograr el desarrollo rápido de aplicaciones que actúan de interfaz con los usuarios. En los últimos años, la red web se ha transformado en la interfaz de usuario con las bases de datos más utilizada.

En este capítulo se estudian las herramientas y las tecnologías necesarias para crear aplicaciones de bases de datos. En concreto, se centrará la atención en las herramientas interactivas que utilizan bases de datos para guardar datos.

Tras una introducción a los programas de aplicación y a las interfaces de usuario, en la Sección 9.1 se tratará el desarrollo de aplicaciones con interfaces basadas en web. Se comienza con una descripción general de las tecnologías web, en la Sección 9.2, y en la Sección 9.3 se tratará la tecnología Java Servlets, que se usa extensamente para la construcción de aplicaciones web. En la Sección 9.4 se presenta una breve introducción a las arquitecturas de aplicaciones web. En la Sección 9.5 se tratan las herramientas para el desarrollo rápido de aplicaciones, y en la Sección 9.6 se ven los temas de rendimiento en la construcción de grandes aplicaciones web. En la Sección 9.7 se tratan los temas de seguridad de las aplicaciones. Se termina el capítulo con la Sección 9.8, en la que se verán los temas de cifrado y su uso en las aplicaciones.

9.1. Interfaces de usuario y programas de aplicación

Aunque mucha gente interactúa con las bases de datos, pocas personas usan un lenguaje de consultas para interactuar directamente con los sistemas de bases de datos. La mayor parte de las personas interactúan con los sistemas de bases de datos usando un programa de aplicación que proporciona una interfaz de usuario, e interacciona con la base de datos por detrás. Estas aplicaciones recogen la entrada de los usuarios, normalmente mediante una interfaz de formularios, e insertan datos en una base de datos o extraen información de la misma de acuerdo con la interacción del usuario, generando una salida que se muestra en la pantalla.

Como ejemplo de aplicación, considere un sistema de matrícula de una universidad. Al igual que otras aplicaciones, el sistema de registro solicitará, en primer lugar, su identidad para realizar la autenticación, normalmente mediante un nombre de usuario y una contraseña. La aplicación utiliza esta información de identificación para extraer la información, como el nombre y asignaturas en las que se esté matriculado, de la base de datos, y mostrará dicha información. La aplicación proporciona un cierto número de interfaces para permitir matricularse en asignaturas y solicitar distintos tipos de información sobre asignaturas y profesores. Las organizaciones utilizan este tipo de aplicaciones para automatizar muchas tareas, como ventas, compras, contabilidad y nóminas, gestión de recursos humanos, gestión de inventario y otras.

Se pueden utilizar programas de aplicación cuando incluso no resulte evidente que se pueden usar. Por ejemplo, un sitio de noticias puede proporcionar una página que se configura de forma transparente para cada usuario, incluso sin que el usuario rellene ningún formulario cuando interacciona con el sitio. Para ello, realmente se ejecuta un programa de aplicación que genera la página personalizada para cada usuario; la personalización puede, por ejemplo, basarse en el histórico de artículos visitados.

Un programa de aplicación típico incluye un componente de «front-end», que trata con la interfaz de usuario, y un componente de «back-end», que se comunica con la base de datos, así como una capa intermedia que contiene la «lógica de negocio», es decir, el código que ejecuta las peticiones concretas de información o actualizaciones; regula las normas del negocio, como las acciones que hay que llevar a cabo para ejecutar una determinada tarea, o quién puede realizar una tarea.

Las arquitecturas de las aplicaciones han evolucionado con el tiempo, como se muestra en la Figura 9.1. Las aflicciones, como las de reserva de vuelos, llevan en funcionamiento desde los años sesenta. En aquellos tiempos, los programas de aplicación se ejecutaban en grandes «mainframe» y los usuarios interactuaban con las aplicaciones mediante terminales, algunos de los cuales disponían de formularios.

Con el amplio uso de los ordenadores personales, muchas organizaciones usaron una arquitectura diferente para las aplicaciones internas, con aplicaciones que se ejecutaban en las computadoras de los usuarios y accedían a una base de datos central. Esta arquitectura, a menudo denominada «cliente-servidor», permitió la creación de interfaces gráficas de usuario muy potentes, que las aplicaciones con terminales anteriores no podían usar. Sin embargo, había que instalar el software en cada una de las máquinas donde se ejecutaba la aplicación, lo que hacía que las actualizaciones fuesen muy costosas. Incluso en la época de las computadoras personales, cuando se hicieron populares las arquitecturas cliente-servidor, las arquitecturas de mainframes continuaron siendo la elección para aplicaciones como las de reservas de vuelos, que se usan desde muchos puntos geográficamente distribuidos.

En los últimos quince años, los navegadores web se han convertido en el *front-end universal* para las aplicaciones de base de datos, conectando con el back-end mediante Internet. Los navegadores utilizan una sintaxis normalizada, el **lenguaje de marcas de hipertexto (HTML),** que admite mostrar información con formato, como la creación de interfaces con formularios. La norma HTML es independiente del sistema operativo o del navegador, y todas las computadoras actuales tienen instalado un navegador. Por tanto, se puede acceder a una aplicación basada en web desde cualquier computadora conectada a Internet.

Al contrario que las arquitecturas cliente-servidor, no es necesario instalar ninguna aplicación específica en la máquina cliente para poder utilizar las aplicaciones basadas en web. Sin embargo, se usan ampliamente interfaces de usuario sofisticadas, con características que van más allá de lo que es posible utilizando únicamente HTML, gracias al lenguaje JavaScript, que se puede utilizar con la mayor parte de los navegadores web. Los programas con JavaScript, al contrario que los escritos en C, se pueden ejecutar en modo seguro, garantizando que no generan problemas de seguridad. Los programas en JavaScript se descargan de forma transparente con el navegador y no necesitan un proceso de instalación de software en la computadora.

Si los navegadores web proporcionan el *front-end* para la interacción con los usuarios, los programas de aplicación constituyen el *back-end*. Normalmente, las peticiones desde un navegador se envían a un servidor web, donde se ejecutan los programas de aplicación que procesan la consulta. Existen diversas tecnologías para la creación de los programas de aplicación que se ejecutan en el back-end, entre otras Java Servlets, Java Server Pages (JSP), Active Server Pages (ASP); o lenguajes de guiones, como PHP, Perl o Python.

En el resto del capítulo se describe cómo se construyen estas aplicaciones, empezando con las tecnologías web y las herramientas para construir interfaces web, así como las tecnologías para construir programas de aplicación, y después se tratarán los temas sobre arquitecturas, rendimiento y seguridad en la creación de aplicaciones.

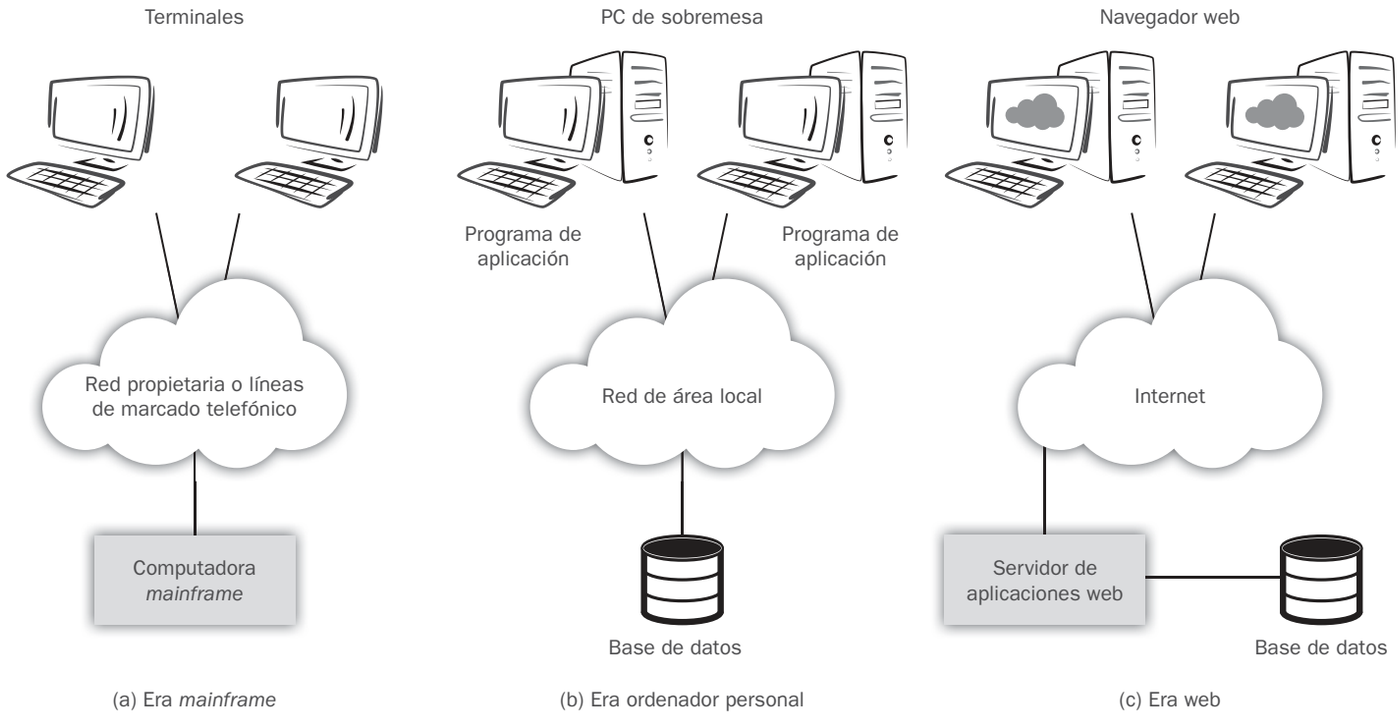


Figura 9.1. Arquitecturas de aplicación en diferente eras.

9.2. Fundamentos de la web

En esta sección se va a repasar parte de la tecnología básica que subyace en la World Wide Web, para los lectores que no estén familiarizados con ella.

9.2.1. Los localizadores uniformes de recursos

Un **localizador uniforme de recursos (URL: *Uniform Resource Locator*)** es un nombre globalmente único para cada documento al que se puede tener acceso en web. Un ejemplo de URL es:

http://www.acm.org/sigmod

La primera parte de la URL[1] indica el modo en que se puede tener acceso al documento: «http» indica que se puede tener acceso al documento mediante el **protocolo de transferencia de hipertexto** (*HyperText Transfer Protocol*: HTTP), que es un protocolo para transferir documentos HTML. La segunda parte indica el nombre único de una máquina con el servidor web. El resto de la URL es el nombre del camino hasta el archivo en la máquina, u otro identificador único del documento dentro de la máquina.

1 A este «localizador», en cuanto «dirección», se le asigna género femenino en el lenguaje coloquial entre usuarios. (*N. del E.*)

Una URL puede contener el identificador de un programa ubicado en la máquina servidora web, así como los argumentos que hay que dar al programa. Un ejemplo de URL de este tipo es:

http://www.google.com/search?q=silberschatz

que dice que el programa search del servidor www.google.com se debe ejecutar con el argumento q=silberschatz. El programa se ejecuta usando el argumento dado, y devuelve un documento HTML que se envía a la interfaz.

```
<html>
<body>
<table border>
<tr> <th>ID</th>       <th>Nombre</th>  <th>Departamento</th> </tr>
<tr> <td>00128</td>  <td>Zhang</td>    <td>Informática</td> </tr>
<tr> <td>12345</td>  <td>Shankar</td> <td>Informática</td> </tr>
<tr> <td>19991</td>  <td>Brandt</td>   <td>Historia</td> </tr>
</table>
</body>
</html>
```

Figura 9.2. Datos tabulares en formato HTML.

9.2.2. El lenguaje de marcas de hipertexto

La Figura 9.2 es un ejemplo fuente de una tabla representado en un documento HTML, mientras la Figura 9.3 muestra la imagen que genera de este documento un navegador a partir de la representación HTML de la tabla. El código fuente en HTML muestra unas pocas etiquetas de HTML. Cada página HTML debe encerrarse con la etiqueta html, y el cuerpo de la página se encierra en una etiqueta body. Una tabla se especifica con la etiqueta table, que contiene las filas que indican las etiquetas tr. La fila de cabecera de la tabla tiene celdas de tabla especificadas con la etiqueta th, mientras que las filas normales se indican con la etiqueta td. No vamos a entrar más en detalle sobre las etiquetas. Véanse las notas bibliográficas para más información sobre la descripción de HTML.

ID	Nombre	Departamento
00128	Zhang	Informática
12345	Shankar	Informática
19991	Brandt	Historia

Figura 9.3. Vista del código HTML de la Figura 9.2.

```
<html>
<body>
<form action="PersonQuery" method=get>
Buscar:
<select name="persontype">
        <option value="student" selected>Estudiante</option>
        <option value="instructor">Profesor</option>
</select> <br>
Nombre: <input type=text size=20 name="name">
<input type=submit value="Enviar">
</form>
</body>
</html>
```

Figura 9.4. Formulario en HTML.

En la Figura 9.4 se muestra cómo especificar un formulario de HTML que permite a los usuarios seleccionar el tipo de persona (estudiante o profesor) en un menú e introducir un número en un cuadro de texto. La Figura 9.5 muestra cómo se muestra el formulario anterior en un navegador web. En el formulario de ejemplo se muestran dos métodos para aceptar entradas del usuario, pero HTML dispone de muchas otras. El atributo action de la etiqueta form especifica que cuando se envíe el formulario, haciendo clic en el botón enviar, los datos del formulario se deben enviar a la URL PersonQuery (la URL es relativa a la de la página). El servidor web está configurado para que cuando se accede a esa URL, se invoca al programa de aplicación correspondiente, con los valores que proporciona el usuario para los argumentos persontype y name (especificados en los campos select e input). El programa de aplicación genera un documento en HTML que se envía de vuelta y se muestra al usuario; más adelante en este capítulo se verá cómo construir estos programas.

HTTP define dos formas para que los valores que introduzca el usuario en el navegador web se envíen al servidor web. El método get codifica los valores formando parte de la URL. Por ejemplo, si la búsqueda de Google usa un formulario con un parámetro de entrada de nombre q con el método get, y el usuario escribe en él la cadena de texto «silberschatz» y envía el formulario, el navegador solicitará la siguiente URL al servidor web:

http://www.google.com/search?q=silberschatz

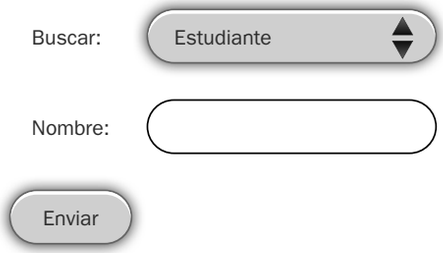


Figura 9.5. Vista del código HTML de la Figura 9.4.

El método post, por el contrario, envía una petición de la URL http://www.google.com, y manda el valor de los parámetros como parte del intercambio del protocolo HTTP entre el servidor web y el navegador. El formulario de la Figura 9.4 especifica que el formulario utiliza el método get.

Aunque el código HTML se puede crear usando un editor de texto sencillo, hay varios editores que permiten la creación directa de texto HTML usando una interfaz gráfica. Estos editores permiten insertar construcciones como los formularios, los menús y las tablas en el documento HTML a partir de un menú de opciones, en lugar de escribir manualmente el código para generar esas construcciones.

HTML soporta *hojas de estilo*, que pueden modificar las definiciones predeterminadas del modo en que se muestran las estructuras de formato HTML, así como otros atributos de visualización como el color de fondo de la página. La norma *hojas de estilo en cascada* (*cascading stylesheet*, CSS) permite usar varias veces la misma hoja de estilo para varios documentos HTML, dando un aspecto uniforme, aunque distintivo, a todas las páginas del sitio web.

9.2.3. Los servidores web y las sesiones

Los **servidores web** son programas que se ejecutan en la máquina servidora, que aceptan las solicitudes de los navegadores web y devuelven los resultados en forma de documentos HTML. El navegador y el servidor web se comunican mediante un protocolo HTTP. Los servidores web proporcionan características potentes, aparte de la mera transferencia de documentos. La característica más importante es la posibilidad de ejecutar programas, con los argumentos proporcionados por el usuario, y devolver los resultados como documentos HTML.

En consecuencia, los servidores web pueden actuar con facilidad como intermediarios para ofrecer acceso a gran variedad de servicios de información. Se pueden crear servicios nuevos mediante la creación e instalación de programas de aplicaciones que los ofrezcan. La norma **interfaz de pasarela común** (*Common Gateway Interface*: CGI) define el modo en que el servidor web se comunica con los programas de las aplicaciones. Los programas de las aplicaciones suelen comunicarse con servidores de bases de datos mediante ODBC, JDBC u otros protocolos, con objeto de obtener o guardar datos.

La Figura 9.6 muestra un servicio web que usa una arquitectura de tres capas, con un servidor web, un servidor de aplicaciones y un servidor de bases de datos. El uso de varios niveles de servidores aumenta la sobrecarga del sistema; la interfaz CGI inicia un nuevo proceso para atender cada solicitud, lo cual supone una sobrecarga aún mayor.

La mayor parte de los servicios web de hoy en día usan una arquitectura web de dos capas, en la que los programas de las aplicaciones se ejecutan en el servidor web, como en la Figura 9.7. En las secciones siguientes se estudiarán con más detalle los sistemas basados en la arquitectura de dos capas.

No existe ninguna conexión continua entre los clientes y los servidores web; cuando un servidor web recibe una solicitud, se crea temporalmente una conexión para enviar la solicitud y recibir la respuesta del servidor web. Pero se cierra la conexión, y la siguiente solicitud llega mediante otra nueva. A diferencia de esto, cuando un usuario inicia una sesión en una computadora, o se conecta a una base de datos mediante ODBC o JDBC, se crea una sesión y en el servidor y en el cliente se conserva la información de la sesión hasta que esta concluye; información como el identificador de usuario y las opciones de sesión que este haya definido. Una razón de peso para que HTTP sea **sin conexión** es que la mayor parte de las computadoras presentan un número limitado de conexiones simultáneas que pueden aceptar, por lo que se podría superar ese límite y denegar el servicio a otros usuarios.

Con un servicio sin conexión, esta se interrumpe en cuanto se satisface una solicitud, lo que deja las conexiones disponibles para otras solicitudes.[2]

La mayor parte de las aplicaciones web necesitan información de sesión para permitir una interacción significativa con el usuario. Por ejemplo, los servicios suelen restringir el acceso a la información y, por tanto, necesitan autenticar a los usuarios. La autenticación debe hacerse una vez por sesión, y las interacciones posteriores de la sesión no deben exigir que se repita la autenticación.

Para implementar sesiones a pesar del cierre de las conexiones, hay que almacenar información adicional en el cliente y devolverla con cada solicitud de la sesión; el servidor usa esta información para identificar que la solicitud forma parte de la sesión del usuario. En el servidor también hay que guardar información adicional sobre la sesión.

Esta información adicional se suele conservar en el cliente en forma de **cookie;** una cookie no es más que un pequeño fragmento de texto, con un nombre asociado, que contiene información de identificación. Por ejemplo, google.com puede definir una cookie con el nombre prefs que codifique las preferencias definidas por el usuario, como el idioma elegido y el número de respuestas mostradas por página. En cada solicitud de búsqueda google.com puede recuperar la cookie denominada prefs del navegador del usuario y mostrar el resultado de acuerdo con las preferencias especificadas. Solo se permite a cada dominio (sitio web) que recupere las cookies que ha definido, no las definidas por otros dominios, por lo que los nombres se pueden reutilizar en otros dominios.

Con objeto de realizar un seguimiento de las sesiones de usuario, una aplicación puede generar un identificador de sesión (generalmente un número aleatorio que no se esté usando como identificador de sesión) y enviar una cookie denominada, por ejemplo, idsesión que contenga ese identificador de sesión. El identificador de sesión también se almacena localmente en el servidor. Cuando llega una solicitud, el servidor de aplicaciones solicita al cliente la cookie denominada idsesión. Si el cliente no posee la cookie almacenada, o devuelve un valor que no se halla registrado como identificador de sesión válido en el servidor, la aplicación concluye que la solicitud no forma parte de una sesión actual. Si el valor de la cookie coincide con el de un identificador de sesión almacenado, la solicitud se identifica como parte de una sesión abierta.

2 Por razones de rendimiento, las conexiones se pueden mantener abiertas durante un momento para permitir más peticiones o reusar la conexión. Sin embargo, no se garantiza que la conexión se mantenga abierta, y las aplicaciones deben diseñarse suponiendo que las conexiones se cierran en cuanto se sirve la petición.

Si una aplicación necesita identificar de manera segura a los usuarios, solo puede definir la cookie después de autentificar al usuario; por ejemplo, se puede autentificar al usuario solo cuando se hayan enviado un nombre de usuario y una contraseña válidos.[3]

Para las aplicaciones que no exigen un nivel elevado de seguridad, como los sitios de noticias abiertos al público, las cookies se pueden almacenar de manera permanente en el navegador y en el servidor; identifican al usuario en visitas posteriores al mismo sitio, sin necesidad de que escriba ninguna información de identificación. Para las aplicaciones que exijan un nivel de seguridad más elevado, el servidor puede invalidar (eliminar) la sesión tras un periodo de inactividad o cuando el usuario la cierre (generalmente los usuarios cierran la sesión pulsando un botón de cierre de sesión, que remite un formulario de cierre de sesión, que invalida la sesión actual). La invalidación de la sesión consiste simplemente en eliminar el identificador de la sesión de la lista de sesiones activas del servidor de aplicaciones.

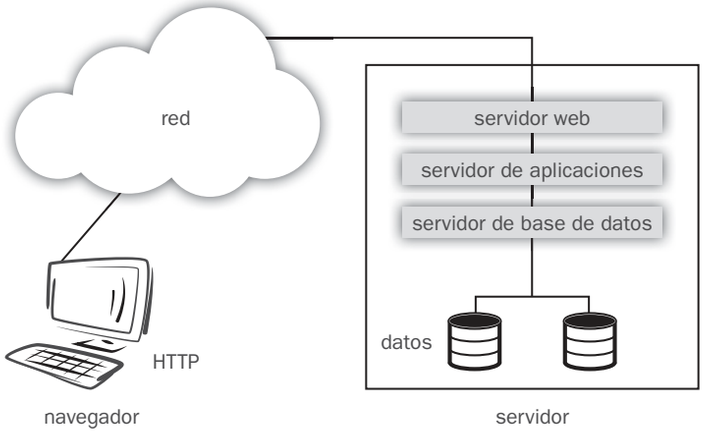


Figura 9.6. Arquitectura de aplicación web de tres capas.

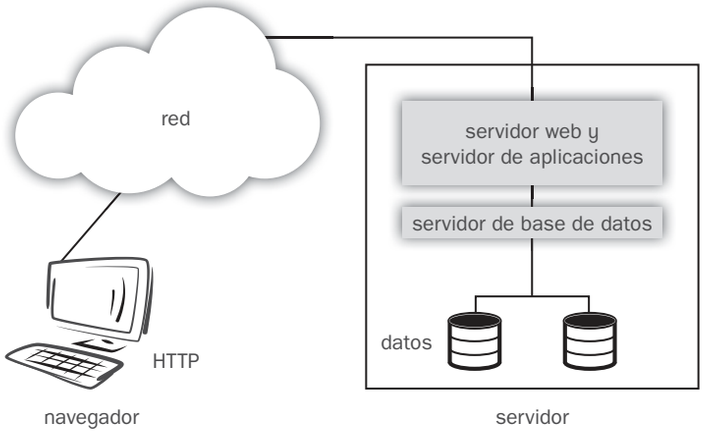


Figura 9.7. Arquitectura de aplicación web en dos capas.

3 El identificador de usuario puede guardarse en la parte del cliente, en una cookie denominada, por ejemplo, idusuario. Estas cookies se pueden usar para aplicaciones con un nivel de seguridad bajo, como los sitios web gratuitos que identifican a sus usuarios. No obstante, para las aplicaciones que exigen un nivel de seguridad más elevado, este mecanismo genera un riesgo: los usuarios malintencionados pueden modificar el valor de las cookies en el navegador y luego suplantar a otros usuarios. La definición de una cookie (denominada idsesión, por ejemplo) con un identificador de sesión generado aleatoriamente (a partir de un espacio de números de gran tamaño) hace muy improbable que ningún usuario pueda suplantar a un usuario diferente (es decir, que pretenda ser otro usuario). Los identificadores de sesión generados de manera secuencial, por otro lado, sí son susceptibles de suplantación.

9.3. Servlets y JSP

En la arquitectura web de dos capas, las aplicaciones se ejecutan como parte del propio servidor web. Un modo de implementar esta arquitectura es cargar los programas Java en el servidor web. La especificación **servlet** de Java define una interfaz de programación de aplicaciones para la comunicación entre el servidor web y los programas de aplicaciones. La clase HttpServlet de Java implementa la especificación de la API servlet; las clases servlet usadas para implementar funciones concretas se definen como subclases de esta clase.[4] A menudo se usa la palabra *servlet* para hacer referencia a un programa (y a una clase) de Java que implementa la interfaz servlet. La Figura 9.8 muestra un ejemplo de código servlet; en breve se explicará con detalle.

El código del servlet se carga en el servidor web cuando se inicia el servidor, o cuando el servidor recibe una solicitud HTTP remota para ejecutar un servlet concreto. La tarea del servlet es procesar esa solicitud, lo cual puede suponer acceder a una base de datos para recuperar la información necesaria, y generar dinámicamente una página HTML para devolvérsela al navegador del cliente.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PersonQueryServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HEAD><TITLE>Resultado de la consulta</TITLE>
                  </HEAD>");
      out.println("<BODY>");

      String persontype = request.getParameter("persontype");
      String number = request.getParameter("name");
      if(persontype.equals("Estudiante")) {
         ... código para encontrar estudiantes con el nombre dado ...
         ... usando JDBC para comunicarse con la base de datos ...
         out.println("<table BORDER COLS=3>");
         out.println("<tr> <td>ID</td> <td>Nombre:</td>" +
                     "<td>Departamento</td> </tr>");
         for(... cada resultado ...) {
            ... obtener ID, nombre y nombre_dept
            ... en variables ID, nombre y nombredept
            out.println("<tr> <td>" + ID + "</td>" + "<td>" + nombre +
                        "</td>" + "<td>" + nombredept + "</td> </tr>");
         }
         out.println("</table>");
      } else {
         ... como antes, pero para profesores ...
      }
      out.println("</BODY>");
      out.close();
   }
}
```

Figura 9.8. Ejemplo de código de un servlet.

4 La interfaz servlet también puede admitir peticiones que no sean HTTP, aunque en el ejemplo solo se usa HTTP.

9.3.1. Un ejemplo de servlet

Los servlets se emplean a menudo para generar de manera dinámica respuestas a las solicitudes de HTTP. Pueden tener acceso a datos proporcionados mediante formularios HTML, aplicar la «lógica de negocio» para decidir la respuesta que deben dar y generar el resultado HTML que se devolverá al navegador.

La Figura 9.8 muestra un ejemplo de código de servlet para implementar el formulario de la Figura 9.4. El servlet se denomina PersonQueryServlet, mientras que el formulario especifica que «action="PersonQuery"». Se debe indicar al servidor web que este servlet se va a utilizar para tratar las solicitudes de PersonQuery. El formulario especifica que se usa el mecanismo get de HTTP para transmitir los parámetros. Por tanto, se invoca el método doGet() del servlet, que se define en el código.

Cada solicitud da lugar a una nueva hebra en la que se ejecuta la llamada, por lo que se pueden tratar en paralelo varias solicitudes. Los valores de los menús del formulario y de los campos de entrada de la página web, así como las cookies, se comunican a un objeto de la clase HttpServletRequest que se crea para la solicitud, y la respuesta a la solicitud se comunica a un objeto de la clase HttpServletResponse.

El método doGet() del ejemplo extrae los tipos de parámetros y el número de parámetros mediante request.getParameter(), y los utiliza para realizar una consulta a la base de datos. El código para tener acceso a la base de datos y obtener los valores de los atributos no se muestra; consulte la Sección 5.1.1.4 para conseguir detalles del modo de uso de JDBC para acceder a una base de datos. El código del servlet devuelve el resultado de la consulta al solicitante imprimiéndolos en formato HTML en la respuesta del objeto HttpServletResponse. La salida de los resultados se implementa creando un objeto out de la clase PrintWriter a partir de response y, después, escribiendo el resultado en formato HTML utilizando out.

9.3.2. Sesiones de los servlets

Recuerde que la interacción entre el navegador y el servidor web carece de estado. Es decir, cada vez que el navegador formula una solicitud al servidor, necesita conectarse al servidor, solicitar alguna información y desconectarse del servidor. Se pueden emplear cookies para reconocer que una solicitud dada procede de la misma sesión de navegador que otra anterior. Sin embargo, las cookies constituyen un mecanismo de bajo nivel y los programadores necesitan una abstracción mejor para tratar con las sesiones.

La API de servlet ofrece un método para realizar el seguimiento de las sesiones y almacenar la información relacionada con ellas. La invocación del método getSession(false) de la clase HttpServletRequest recupera el objeto HttpSession correspondiente al navegador que envió la solicitud. Si el valor del argumento se pone a true significa que hay que crear un nuevo objeto sesión si la solicitud fuera nueva.

Cuando se invoca el método getSession(), el servidor pide primero al cliente que devuelva una cookie con un nombre concreto. Si el cliente no tiene ninguna cookie con ese nombre, o devuelve un valor que no corresponde al de ninguna sesión abierta, la solicitud no forma parte de ninguna sesión abierta. En ese caso, devolverá un valor null y el servlet puede dirigir al usuario a una página de identificación.

La página de identificación permite que el usuario introduzca su nombre de usuario y su contraseña. El servlet correspondiente a la página de identificación puede comprobar que la contraseña coincide con la del usuario (por ejemplo, buscando la información de autenticación en la base de datos). Si el usuario se identifica correctamente, el servlet de identificación ejecuta getSession(true), que devuelve un nuevo objeto sesión. Para crear una nueva sesión,

el servidor web ejecuta internamente las siguientes tareas: definir una cookie (denominada, por ejemplo, idsesión) con un identificador de sesión como valor asociado en el navegador cliente, crear un nuevo objeto sesión y asociar el valor del identificador de sesión con el objeto sesión.

El código del servlet también puede almacenar y buscar pares (nombre-atributo, valor) en el objeto HttpSession, para mantener el estado entre varias solicitudes de una misma sesión. Por ejemplo, el servlet del inicio de sesión puede almacenar el identificador de usuario como parámetro de la sesión ejecutando el método

```
sesion.setAttribute("idusuario", idusuario)
```

sobre el objeto sesión que devuelve getSession(); se supone que la variable de Java idusuario contiene el identificador del usuario.

Si la solicitud formaba parte de una sesión abierta, el navegador habría devuelto el valor de la cookie y el servidor web habría devuelto el objeto sesión correspondiente. El servlet puede recuperar entonces los parámetros de la sesión, como el identificador de usuario, a partir del objeto sesión ejecutando el método:

```
sesion.getAttribute("idusuario")
```

con el objeto sesión obtenido anteriormente. Si el atributo idusuario no está definido, la función devuelve el valor nulo, lo que indica que el usuario del cliente no se ha autenticado.

9.3.3. Ciclo de vida de los servlets

El ciclo de vida de cada servlet está controlado por el servidor web en el que se ha desplegado. Cuando hay una solicitud de un cliente para un servlet concreto, el servidor comprueba primero si existe algún ejemplar de ese servlet o no. Si no existe, el servidor web carga la clase del servlet en la máquina virtual de Java (Java virtual machine, JVM) y crea un ejemplar de la clase del servlet. Además, el servidor llama al método init() para inicializar el ejemplar del servlet. Tenga en cuenta que cada ejemplar del servlet solo se inicializa una vez, cuando se carga.

Tras asegurarse de que existe el ejemplar del servlet, el servidor invoca el método service() del servlet, con un objeto request y un objeto response como parámetros. De manera predeterminada, el servidor crea una hebra nueva para ejecutar el método service(); por tanto, se pueden ejecutar en paralelo varias solicitudes al servlet, sin tener que esperar que solicitudes anteriores completen su ejecución. El método service() llama a doGet() o a doPost(), según corresponda.

Cuando ya no sea necesario, se puede cerrar el servlet llamando al método destroy(). El servidor se puede configurar para que cierre de manera automática el servlet si no se han hecho solicitudes en un periodo de tiempo determinado; este periodo es un parámetro del servidor que se puede definir según se considere adecuado para la aplicación.

9.3.4. Soporte de los servlets

Muchos servidores de aplicaciones ofrecen soporte predeterminado para servlets. Entre los más utilizados está Tomcat Server del Apache Jakarta Project. Otros servidores de aplicaciones que admiten el uso de servlet son Glassfish, JBoss, **BEA** Weblogic Application Server, Oracle Application Server y WebSphere Application Server de IBM.

La forma más apropiada de desarrollar aplicaciones con servlet es utilizar un **IDE** como Eclipse o NetBeans, que vienen con servidores Glassfish o Tomcat incorporados.

Estos servidores de aplicaciones ofrecen gran variedad de servicios adicionales, aparte del soporte básico para servlets. Permiten desplegar aplicaciones y pararlas, así como proporcionar funciones para monitorizar el estado del servidor de aplicaciones, incluyendo estadísticas de rendimiento. Si se modifica el código de un servlet, pueden detectarlo y volver a compilar y a cargar el servlet de manera transparente. Muchos servidores de aplicaciones permiten que el servidor se ejecute en varias máquinas en paralelo para mejorar el rendimiento y encaminar las solicitudes a la copia adecuada. Muchos servidores de aplicaciones soportan también la plataforma Java 2 Enterprise Edition (J2EE), que ofrece soporte y API para gran variedad de tareas, como el manejo de objetos, el procesamiento en paralelo en varios servidores de aplicaciones y el manejo de datos en XML (XML se describe más adelante, en el Capítulo 23).

9.3.5. Secuencias de comandos en el lado del servidor

La escritura, incluso de una mera aplicación web, en un lenguaje de programación como Java o C supone una tarea que consume bastante tiempo y necesita muchas líneas de código y programadores familiarizados con las complejidades del lenguaje. Un enfoque alternativo, el de las **secuencias de comandos en el lado del servidor,** ofrece un método mucho más sencillo para la creación de múltiples aplicaciones. Los lenguajes de guiones ofrecen estructuras que pueden incluirse en los documentos HTML. En los guiones en el lado del servidor, antes de entregar una página web, el servidor ejecuta las secuencias incluidas en el contenido de la página en HTML. Cada secuencia, al ejecutarse, puede generar texto que se añade a la página (o que incluso puede eliminar contenido de la página). El código fuente de los guiones se elimina de la página, de modo que puede que el cliente ni siquiera se dé cuenta de que la página contenía originalmente algún código. Puede que el guion ejecutado contenga un código SQL que se ejecute usando una base de datos.

Entre los lenguajes de guiones más utilizados se encuentran: Java Server Pages (JSP) de Sun, Active Server Pages (ASP) y su sucesor ASP.NET de Microsoft, PHP (PHP Hypertext Preprocessor, PHP), el lenguaje de marcas de ColdFusion (ColdFusion Markup Language, CFML) y Ruby on Rails. Muchos lenguajes de guiones también permiten incluir un código en lenguajes como Java, C#, VBScript, Perl y Python, de forma que se incluya en el HTML o se invoque desde páginas HTML. Por ejemplo, JSP permite incluir código en Java en las páginas en HTML, mientras que ASP.NET de Microsoft y ASP permiten incluir C# y VBScript. Muchos de estos lenguajes disponen de bibliotecas y herramientas que constituyen un marco («framework») para el desarrollo de aplicaciones web.

```
<html>
<head> <title>Hola</title> </head>
<body>
  <% if (request.getParameter("nombre") == null)
    {out.println("Hola mundo"); }
    else {out.println("Hola, " + request.getParameter("nombre")); }
  %>
</body>
</html>
```

Figura 9.9. Página en JSP con código Java incrustado.

A continuación se describe brevemente **Java Server Pages** (**JSP**), un lenguaje de guiones que permite que los programadores de HTML mezclen HTML estático con el generado de manera dinámica. El motivo es que, en muchas páginas web dinámicas, la mayor parte del contenido sigue siendo estático (es decir, se halla presente el mismo contenido siempre que se genera la página). El contenido dinámico de la página web (que se genera, por ejemplo, en función de los parámetros de los formularios) suele ser una pequeña parte de la página. La creación de estas páginas mediante la escritura de código servlet da lugar a grandes cantidades de HTML que se codifican como cadenas de Java. Por el contrario, JSP permite que el código Java se incorpore al HTML estático; el código Java incorporado genera la parte dinámica de la página. Los guiones de JSP realmente se traducen en código de servlet que luego se compila, pero se le ahorra al programador de la aplicación el problema de tener que escribir gran parte del código para crear el servlet.

La Figura 9.9 muestra el texto fuente de una página JSP que incluye un código Java incrustado. El código Java se distingue del de HTML circundante por estar encerrado entre <% … %>. El código usa request.getParameter() para obtener el valor del atributo nombre.

Cuando el navegador solicita una página JSP, el servidor de aplicaciones genera la salida en HTML de la página, que se envía de vuelta al navegador. La parte en HTML de la página en JSP se genera tal cual.[5] Siempre que haya código Java incluido entre <% … %>, el código se sustituye en la salida en HTML por el texto que escribe el objeto out. En el código JSP de la figura anterior; si no se introduce ningún valor en el parámetro nombre del formulario, el guion escribe: «Hola mundo»; si se introduce un valor, el guion escribe «Hola» seguido del nombre escrito.

Un ejemplo más realista puede llevar a cabo acciones más complejas, como buscar valores en una base de datos usando JDBC.

JSP también dispone del concepto de *biblioteca de etiquetas* (*tag library*), que permite el uso de etiquetas que se parecen mucho a las etiquetas HTML, pero se interceptan en el servidor y se sustituyen por el código HTML correspondiente. JSP ofrece un conjunto estándar de etiquetas que definen variables y controlan el flujo (iteradores, si-entonces-sino), junto con un lenguaje de expresiones basado en Javascript (pero interpretado en el servidor). El conjunto de etiquetas es extensible, y ya se han implementado varias bibliotecas de etiquetas. Por ejemplo, existe una biblioteca de etiquetas que soporta la visualización paginada de conjuntos de datos de gran tamaño y una biblioteca que simplifica la visualización y el análisis de fechas y de horas. Véanse las notas bibliográficas para conocer más referencias de información sobre las bibliotecas de etiquetas de JSP.

9.3.6. Secuencias de comandos en el lado del cliente

La inclusión de códigos de programas en los documentos permite que las páginas web sean **activas,** logrando actividades como animaciones ejecutando programas en el lado local, en lugar de presentar únicamente texto y gráficos pasivos. El uso principal de estos programas es una interacción flexible con el usuario, más allá de la limitada interacción de HTML y sus formularios. Más aún, la ejecución de los programas en el lado del cliente acelera la interacción en gran medida, comparado con la interacción de enviar los elementos al lado del servidor para su procesamiento.

5 JSP permite incluir código de forma más compleja, donde el código HTML se encuentra en una sentencia if-else (si-sino), y genera la salida de forma condicional dependiendo de si la condición se evalúa a cierto o falso.

PHP

PHP es un lenguaje de guiones en el lado del servidor muy utilizado. El código en PHP se puede mezclar con el código en HTML, de forma similar a JSP. Los caracteres «<?php» indican el comienzo de código en PHP, y los caracteres «?>» indican el fin del código en PHP. El siguiente ejemplo realiza las mismas acciones que el código JSP de la Figura 9.9.

```
<html>
<head> <title> Hola </title> </head>
<body>
  <?php if (!isset($ REQUEST['nombre']))
   { echo 'Hola mundo; }
   else { echo 'Hola, ' . $ REQUEST['nombre']; }
  ?>
</body>
</html>
```

El array $_REQUEST contiene los parámetros de la petición. Fíjese que el array usa como índice el parámetro nombre; en PHP los arrays se pueden indicar con cadenas arbitrarias y no solo con números. La función isset comprueba si el elemento del array está inicializado. La función echo escribe sus argumentos en la salida de HTML. El operador «.» entre las dos cadenas de caracteres concatena las cadenas de texto.

Un servidor web configurado apropiadamente interpreta cualquier archivo cuyo nombre termine en «.php» como un archivo PHP. Si se solicita este archivo, el servidor web lo procesa de forma similar a como se procesan los archivos JSP, y devuelve el código HTML generado al navegador.

Existen distintas bibliotecas disponibles para el lenguaje PHP, incluyendo bibliotecas para el acceso a bases de datos usando ODBC (similar a JDBC en Java).

Un peligro de este tipo de programas es que si el diseño del sistema no se ha realizado cuidadosamente, el código incluido en la página web (o equivalente en un mensaje de email), puede realizar acciones maliciosas en la computadora del usuario. Estas acciones maliciosas pueden ir desde leer información privada o borrar o modificar información de la computadora, hasta tomar el control de la computadora y propagar el código a otras computadoras (a través de email, por ejemplo). Varios virus de correo se han extendido ampliamente en los últimos años de esta forma.

Una de las razones de que el lenguaje Java se haya popularizado es que proporciona un modo seguro para la ejecución de programas en la computadora del usuario. El código en Java se puede compilar en una plataforma de «byte-code» independiente que se puede ejecutar en cualquier navegador que admita Java. Al contrario que otros programas, los programas Java (applets) se descargan como parte de la página web y no tienen capacidad de realizar ninguna acción que pueda ser destructiva. Se les permite mostrar datos en la pantalla o realizar una conexión de red al servidor desde donde se descargó la página web para obtener más información. Sin embargo, no se les permite el acceso a archivos locales, ni ejecutar programas del sistema, ni realizar conexiones de red a otras computadoras.

Mientras que Java es un lenguaje de programación completo, los hay más sencillos, llamados **lenguajes de guiones,** que pueden enriquecer la interacción del usuario y proporcionar, a la vez, la misma protección que Java. Estos lenguajes ofrecen construcciones que se pueden incluir en un documento HTML. Los **lenguajes de secuencias de comandos en el lado del cliente** son lenguajes diseñados para su ejecución en el navegador web del cliente.

De ellos, el lenguaje *JavaScript* es con mucho el más utilizado. La generación actual de interfaces web usa el lenguaje JavaScript intensivamente para construir interfaces de usuario sofisticadas. JavaScript se usa para múltiples tareas. Por ejemplo, se pueden usar funciones para comprobar errores (validación) de las entradas del usuario, como si una fecha tiene el formato apropiado, o si un determinado valor (como la edad) tiene un valor adecuado. Estas comprobaciones se realizan antes de enviar los datos al servidor web.

La Figura 9.10 muestra un ejemplo de función JavaScript utilizada para validar una entrada de formulario. La función se declara en la sección head del documento HTML. La función comprueba que los créditos introducidos para una asignatura es un número mayor que 0, y menor que 16. La etiqueta form especifica la función de validación que hay que invocar cuando el usuario envíe el formulario. Si la validación falla, se muestra un cuadro de alerta al usuario, y si es correcta el formulario se envía al servidor.

Se puede utilizar JavaScript para modificar dinámicamente el código HTML que se muestra. El navegador analiza el código HTML en la memoria en una estructura de árbol definida en una norma que se denomina **Modelo de Objetos de Documento** (DOM: *Document Object Model*). El código JavaScript puede modificar la estructura en árbol para llevar a cabo ciertas operaciones. Por ejemplo, suponga que un usuario necesita introducir un cierto número de datos, por ejemplo, varios elementos en una factura. Se puede utilizar para recoger la información una tabla que contenga cuadros de texto u otros métodos de entrada. La tabla puede tener un tamaño predeterminado, pero si se necesitan más filas, el usuario puede pulsar en un botón con la etiqueta, por ejemplo, «Añadir elemento». Este botón puede estar configurado para invocar a una función en JavaScript que modifica el árbol DOM para añadir una fila extra en la tabla.

Aunque el lenguaje JavaScript está normalizado, existen diferencias entre distintos navegadores, en particular en los detalles del modelo DOM. El resultado es que el código JavaScript que funciona en un navegador puede no funcionar en otro. Para evitar estos problemas, es mejor utilizar bibliotecas de JavaScript, como la biblioteca YUI de Yahoo, que permite escribir el código de forma independiente del navegador. Internamente, las funciones de la biblioteca pueden descubrir qué navegador se está utilizando y utilizar el código apropiado en el navegador. Véase la sección Herramientas al final del capítulo para más información sobre YUI y otras bibliotecas.

En la actualidad, JavaScript se utiliza ampliamente para crear páginas web dinámicas, junto con varias tecnologías que colectivamente se denominan **Ajax.** Los programas escritos en JavaScript se comunican con el servidor web de forma asíncrona, es decir, en segundo plano, sin bloquear la interacción del usuario con el navegador web, y pueden obtener información y mostrarla.

Como ejemplo de uso de Ajax, considere un sitio web con un formulario que permita seleccionar un país y, una vez seleccionado, se pueda elegir la provincia o estado de una lista de provincias o estados del país. Hasta que se seleccione el país, el desplegable de provincias está vacío. Ajax permite que la lista de provincias se descargue del servidor web en segundo plano, mientras se selecciona el país, y en cuanto la lista se ha conseguido, se añade a la lista desplegable y le permite al usuario seleccionar la provincia.

También existen lenguajes de secuencias de comandos de propósito especial para tareas especializadas, como las animaciones (por ejemplo, Flash y Shockwave) y el modelado en tres dimensiones (Lenguaje de Marcado de Realidad Virtual, VRML, Virtual Reality Markup Language). Flash se usa extensamente en la actualidad no solo para las animaciones, sino también para vídeo en streaming.

```
<html>
<head>
<script type="text/javascript">
  function validate() {
    var credits=document.getElementById("credits").value;
    if (isNaN(credits)|| credits<=0 || credits>=16) {
      alert("Los créditos debe ser un valor mayor que 0
      y menor que 16");
      return false
    }
  }
</script>
</head>

<body>
<form action="createCourse" onsubmit="return validate()">
  Título: <input type="text" id="title" size="20"><br />
  Créditos: <input type="text" id="credits" size="2"><br />
  <input type="submit" value="Enviar">
</form>
</body>
</html>
```

Figura 9.10. Ejemplo de JavaScript usado para validar un formulario de entrada de datos.

9.4. Arquitecturas de aplicación

Para gestionar la complejidad, las grandes aplicaciones se suelen dividir en varias capas:

- La capa de *presentación* o *interfaz de usuario*, que trata de la interacción con el usuario. Una misma aplicación puede tener distintas versiones de esta capa, que se corresponden con los distintos tipos de interfaces, como navegadores web e interfaces de usuario en teléfonos móviles, que tienen pantallas más pequeñas.

  En muchas implementaciones, la capa de presentación/interfaz de usuario se divide conceptualmente en capas, de acuerdo con la arquitectura **modelo-vista-controlador** (MVC). El **modelo** se corresponde con la capa de lógica de negocio, que se describe a continuación. La **vista** se corresponde con la presentación de los datos, un mismo modelo subyacente puede tener distintas vistas dependiendo del software o dispositivo que se use para acceder a la aplicación. El **controlador** recibe eventos (acciones del usuario), ejecuta acciones del modelo y devuelve una vista al usuario. La arquitectura MVC se usa en gran número de «frameworks» de aplicaciones web, que se tratan más adelante en la Sección 9.5.2.
- La capa de la **lógica de negocio** proporciona una vista de alto nivel y acciones con los datos. Se trata esta capa con más detalle en la Sección 9.4.1.
- La capa de **acceso a los datos** proporciona la interfaz entre la capa de la lógica de negocio y la base de datos subyacente. Muchas aplicaciones usan un lenguaje orientado a objetos para codificar la capa de la lógica de negocio, y usan un modelo de datos orientado a objetos, siendo la base de datos subyacente una base de datos relacional. En estos casos, la capa de acceso a los datos también proporciona una correspondencia entre el modelo de datos orientado a objetos que usa la lógica de negocio y el modelo relacional que usa la base de datos. Se trata esta correspondencia con más detalle en la Sección 9.4.2.

La Figura 9.11 muestra las capas citadas, junto con una secuencia de pasos del proceso de una solicitud desde el navegador web. Las etiquetas en las flechas de la figura indican el orden de los pasos. Cuando se recibe una solicitud del servidor de aplicaciones, el controlador envía una petición al modelo. El modelo procesa la solicitud, usando la lógica de negocio, que puede implicar la actualización de los objetos que forman parte del modelo, seguido de la creación de un objeto resultado.

El modelo después usa la capa de acceso a los datos para actualizar u obtener información de la base de datos. El objeto resultado que crea el modelo se envía al módulo de vista, que crea la vista en HTML del resultado, para que se muestre en el navegador web. La vista se puede adaptar de acuerdo con las características del dispositivo que se utilice para mostrar el resultado, por ejemplo, si se utiliza un monitor de computadora con una gran pantalla o una pantalla pequeña en un teléfono.

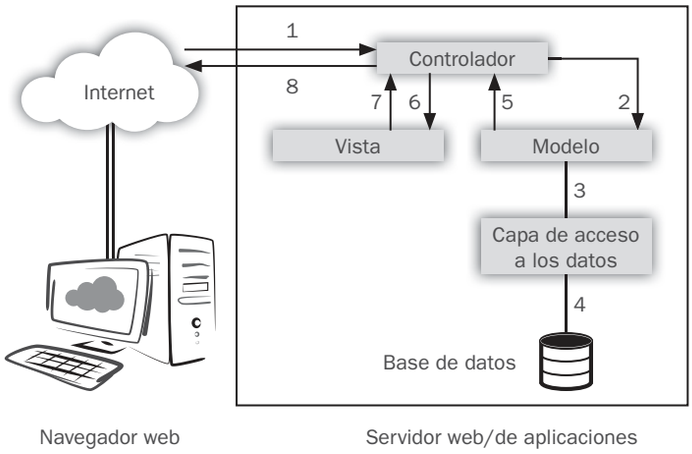


Figura 9.11. Arquitectura de aplicación web.

9.4.1. Capa de la lógica de negocio

La capa de la lógica de negocio de una aplicación para la gestión de una universidad puede proporcionar abstracciones de entidades como estudiantes, profesores, asignaturas, secciones, etc., y acciones como admitir a un estudiante en la universidad, matricular a un estudiante en una asignatura, etc. El código para implementar estas acciones asegura que las **reglas del negocio** se satisfacen; por ejemplo, el código debería asegurar que un estudiante se puede matricular de una asignatura solo si cumple todos los prerrequisitos y ha pagado sus tasas universitarias.

Además, la lógica de negocio incluye **flujos de trabajo** que describen cómo se gestiona una tarea que implica a varios partícipes. Por ejemplo, si un candidato opta a la universidad, existe un flujo de trabajo que define quién debería ver y aprobar la opción y si se aprueba en este primer paso, quién debería ver la opción a continuación, y así sucesivamente hasta que se le ofrece la aceptación al estudiante o se le envía una nota de rechazo. La gestión del flujo de trabajo necesita tratar con situaciones de error; por ejemplo, si no se cumplen los plazos para una aceptación/rechazo, puede que haya que informar a un supervisor para que intervenga y asegure que se procesa. Los flujos de trabajo se tratan con más detalle en la Sección 26.2.

9.4.2. La capa de acceso a los datos y la correspondencia entre objetos y relaciones

En el escenario más simple, en el que la capa de la lógica de negocio usa el mismo modelo de datos que la base de datos, la capa de acceso a los datos simplemente oculta los detalles de la interfaz con la base de datos. Sin embargo, cuando la capa de la lógica de negocios se escribe utilizando un lenguaje orientado a objetos, resulta natural modelar los datos como objetos, con métodos que se invocan sobre los objetos.

En las primeras implementaciones, los programadores tenían que escribir un código para crear los objetos recuperando los datos de la base de datos, así como para guardar las actualizaciones en los objetos de vuelta a la base de datos. Sin embargo, estas conversiones manuales entre los modelos de datos son muy engorrosas y propensas a errores. Una forma de manejar este problema fue desarrollar un sistema de base de datos que guardaba de forma nativa objetos y relaciones entre los objetos. Estas bases de datos, denominadas **bases de datos orientadas a objetos,** se tratan con más detalle en el Capítulo 22. Sin embargo, estas bases de datos no alcanzaron un gran éxito comercial por varias razones técnicas y comerciales.

Una alternativa es usar bases de datos relacionales tradicionales para guardar los datos, pero para automatizar la correspondencia entre los datos de las relaciones con los objetos en memoria, que se crean bajo demanda (y ya que la memoria normalmente no es suficiente para almacenar todos los datos de la base de datos), así como la correspondencia inversa para guardar los objetos actualizados de nuevo como relaciones en la base de datos.

Se han desarrollado varios sistemas para implementar esta **correspondencia entre objetos y relaciones.** El sistema **Hibernate** se usa extensamente para la correspondencia entre objetos de Java y relaciones. En Hibernate, la correspondencia entre una clase de Java y una o más relaciones se especifica en un archivo de correspondencias. Este archivo puede indicar, por ejemplo, que la clase de Java de nombre *Estudiante* se hace corresponder con la relación *estudiante*, donde el atributo *ID* de Java se hace corresponder al atributo *estudiante.ID*, y así sucesivamente. La información sobre la base de datos, como la máquina en la que se ejecuta, y el nombre de usuario y contraseña para la conexión a la base de datos, etc., se especifican en un archivo de *propiedades.* El programa tiene que abrir una *sesión*, que establece la conexión con la base de datos. Cuando se establece la conexión, un objeto *est* de la clase *Estudiante* creado en Java se puede guardar en la base de datos invocando a `session.save(est)`. El código Hibernate genera los comandos SQL necesarios para guardar los datos correspondientes a *estudiante* en la relación.

Se puede obtener una lista de los objetos de la base de datos ejecutando una consulta escrita en el lenguaje de consultas Hibernate; es similar a ejecutar una consulta usando JDBC, que devuelve un `ResultSet` con un conjunto de tuplas. De forma alternativa, se puede obtener un único objeto proporcionando su clave primaria. Los objetos obtenidos se pueden modificar en memoria; cuando la transacción de la sesión de Hibernate sigue activa y se confirma, Hibernate guarda automáticamente los objetos actualizados realizando las correspondientes actualizaciones en las relaciones de la base de datos.

Mientras que las entidades del modelo E-R se corresponden con objetos en un lenguaje orientado a objetos como es Java, las relaciones no. Hibernate tiene la capacidad de hacer corresponder tales relaciones como conjuntos asociados con objetos. Por ejemplo, la relación *matricula* entre *estudiante* y *sección* se puede modelar asociando un conjunto de *secciones* con cada *estudiante* y un conjunto de *estudiantes* con cada *sección.* Una vez especificada la correspondencia, Hibernate puebla estos conjuntos de forma automática a partir de la relación *matricula* de la base de datos, y las actualizaciones en los conjuntos se refleja de vuelta en la base de datos cuando se realiza una confirmación (*commit*).

La característica anterior ayuda a proporcionar al programador un modelo de alto nivel de los datos sin preocuparse por los detalles del almacenamiento relacional. Sin embargo, Hibernate como otros sistemas de correspondencia entre objetos y relacional, también permite que los programadores puedan acceder directamente al SQL de las relaciones subyacentes. Este acceso directo resulta particularmente importante para escribir consultas complejas necesarias en la generación de informes.

Microsoft ha desarrollado un modelo de datos, llamado **Modelo Dato-Entidad,** que se puede ver como una variante del modelo entidad-relación y un «framework» asociado, llamado ADO.NET Entity Framework, que puede hacer la correspondencia entre datos del Modelo Dato-Entidad y la base de datos relacional. El «framework» también proporciona un lenguaje tipo-SQL llamado **Entity SQL,** que funciona directamente sobre el Modelo Datos-Entidad.

EJEMPLO DE HIBERNATE

Como ejemplo del uso de Hibernate, se crea una clase Java de la relación *estudiante* de la siguiente forma:

```
public class Estudiante {
    String ID;
    String nombre;
    String departamento;
    int tot_cred;
    Estudiante(String id, String nombre, String dept, int totcreds);
    // constructor
}
```

Para ser precisos, los atributos de la clase se deberían declarar privados y proporcionar métodos get/set para acceder a los atributos, pero se omiten estos detalles.

La correspondencia de los atributos de la clase Estudiante en atributos de la relación *estudiante* se especifica en el archivo de correspondencias, escrito en XML. De nuevo se omiten los detalles.

El siguiente fragmento de código crea un objeto Estudiante y lo guarda en la base de datos.

```
Session sesion = getSessionFactory().openSession();
Transaction txn = sesion.beginTransaction();
Student est = new Estudiante("12328", "John Smith",
                                "Informática", 0);
sesion.save(est);
txn.commit();
sesion.close();
```

Hibernate genera automáticamente las sentencias **insert** de SQL necesarias para crear una tupla *estudiante* en la base de datos.

Para obtener los estudiantes, se puede usar el siguiente fragmento de código:

```
Session sesion = getSessionFactory().openSession();
Transaction txn = sesion.beginTransaction();
List estudiantes =
 sesion.find("from Estudiante as e order by e.ID asc");
for ( Iterator iter = estudiantes.iterator(); iter.hasNext(); ) {
 Estudiante est = (Estudiante) iter.next();
 … escribir la información del Estudiante …
}
txn.commit();
sesion.close();
```

El fragmento de código anterior usa una consulta en el lenguaje de consulta HQL de Hibernate. La consulta en HQL se traduce automáticamente a SQL y se ejecuta, y el resultado se convierte en una lista de objetos *Estudiante*. El bucle for itera sobre los objetos de esta lista y los escribe por pantalla.

9.4.3. Servicios web

En el pasado, la mayoría de las aplicaciones web solo usaban datos disponibles en el servidor de aplicaciones y su base de datos asociada. Desde hace unos años existe una amplia variedad de datos disponibles en la web que se pretende se procesen por programas, en lugar de mostrarse directamente al usuario; el programa puede estar dándose como parte de una aplicación de back-end, o puede ser un guion ejecutándose en el navegador. A estos datos se accede normalmente utilizando una interfaz de programación de aplicaciones web, es decir, se envía una solicitud de llamada a una función utilizando el protocolo HTTP; se ejecuta en un servidor de aplicaciones y los resultados se envían de vuelta al programa que hizo la llamada. Un sistema con este tipo de interfaz se llama **servicio web.**

Se utilizan dos formas para implementar servicios web. En la forma más sencilla, llamada **transferencia del estado de representación** (REST: *Representation State Transfer*), las llamadas a funciones del servicio web se ejecutan usando una solicitud estándar de HTTP a una URL de un servidor de aplicaciones con parámetros enviados como parámetros estándar de solicitudes de HTTP. El servidor de aplicaciones ejecuta la solicitud, lo que puede implicar actualizar la base de datos en el servidor, genera y codifica el resultado y devuelve este como resultado a una solicitud de HTTP. El servidor puede usar cualquier codificación para una determinada URL solicitada; se usan ampliamente XML y una codificación de objetos de JavaScript denominada **notación JavaScript de objetos** (JSON: *JavaScript Object Notation*). El solicitante analiza la página de respuesta para acceder a los datos devueltos.

En muchas aplicaciones de servicios web RESTful, es decir, servicios web que usan REST, el solicitante es un código en JavaScript ejecutado en un navegador web; el código actualiza la pantalla del navegador usando el resultado de la llamada a la función. Por ejemplo, cuando desea desplazarse por la interfaz de un mapa en la web, la parte del mapa que es necesario mostrar como nueva se puede solicitar desde código en JavaScript usando una interfaz RESTful y después mostrarla en la pantalla.

Una forma más compleja, a veces denominada «Grandes Servicios Web» («Big Web Services»), usa una codificación XML de los parámetros y de los resultados, tiene definiciones formales de la API de web usando un lenguaje especial, y usa una capa de protocolo construida encima del protocolo HTTP. Esta forma se describe con más detalle en la Sección 23.7.3.

9.4.4. Operación desconectado

Muchas aplicaciones necesitan seguir funcionando incluso aunque el cliente esté desconectado del servidor de aplicaciones. Por ejemplo, un estudiante que desea rellenar un formulario aunque su portátil esté desconectado de la red, y poder guardarlo para cuando se vuelva a conectar. Otro ejemplo, si un cliente de correo electrónico se ha construido como aplicación web, el usuario puede desear escribir un correo incluso aunque su portátil este desconectado de la red, y que se envíe cuando vuelva a conectarse. La construcción de este tipo de aplicaciones requiere un almacenamiento local, preferiblemente en forma de base de datos. El software **Gears,** desarrollado inicialmente por Google, es una extensión del navegador que proporciona una base de datos, un servidor web local, y permite la ejecución paralela de JavaScript en el cliente. El software funciona de forma idéntica en distintas plataformas y sistemas operativos, permitiendo a las aplicaciones disponer de una rica funcionalidad sin necesidad de instalar ningún software adicional, distinto del propio Gears. El software AIR de Adobe también proporciona una funcionalidad similar para la construcción de aplicaciones de Internet que se pueden ejecutar fuera de un navegador web.

9.5. Desarrollo rápido de aplicaciones

Si las aplicaciones web se construyen sin utilizar herramientas ni bibliotecas para la generación de interfaces de usuario, el esfuerzo de programación necesario para crear estas interfaces de usuario puede ser significativamente mayor que el que se necesita para la lógica de negocio y el acceso a la base de datos. Se han desarrollado varias estrategias para reducir el esfuerzo requerido para la construcción de aplicaciones:

- Proporcionar una biblioteca de funciones para generar los elementos de la interfaz de usuario con una programación mínima.
- Proporcionar funciones de arrastrar en un entorno integrado de desarrollo que permita arrastrar los elementos de la interfaz de usuario desde un menú a una vista de diseño. El entorno integrado de desarrollo genera el código que crea la interfaz de usuario invocando funciones de biblioteca.
- Generar automáticamente el código de la interfaz de usuario a partir de una especificación declarativa.

Todas estas formas se han utilizado para la creación de interfaces gráficas de usuario, antes de que existiese la web, formando parte de las herramientas de **desarrollo rápido de aplicaciones** (RAD: *Rapid Application Development*) y ahora también se usan ampliamente para la creación de aplicaciones web.

Entre los ejemplos de aplicaciones para el desarrollo rápido de interfaces para aplicaciones de bases de datos se encuentran Oracle Forms, Sybase PowerBuilder y Oracle Application Express (APEX). Además, aplicaciones diseñadas para el desarrollo de aplicaciones web, como Visual Studio y Netbeans VisualWeb, disponen de características para el desarrollo rápido de interfaces web para las aplicaciones con bases de datos.

En la Sección 9.5.1 se tratan las herramientas para la construcción de interfaces de usuario y en la Sección 9.5.2 se ven los «frameworks» que permiten la generación automática del código a partir de un modelo.

9.5.1. Herramientas para la construcción de interfaces de usuario

Muchas construcciones de HTML se generan mejor si se utilizan funciones definidas apropiadamente, en lugar de escribirlas como parte del código de las páginas web. Por ejemplo, los formularios de direcciones suelen requerir un menú con los países y sus provincias o estados. En lugar de escribir cada vez el código HTML necesario para crear este menú, es preferible definir una función que genere el menú y llamarla siempre que se necesite.

Los menús se suelen generar preferiblemente a partir de datos que se encuentran en una base de datos, como una tabla con los nombres de los países y de las provincias o estados. La función que genera el menú ejecuta una consulta a la base de datos y rellena el menú usando el resultado de la consulta. Añadir un país o una provincia o estado solo requiere actualizar la base de datos, y no el código de la aplicación. Tiene la desventaja de requerir una mayor interacción con la base de datos, pero esta sobrecarga se puede minimizar haciendo caché de los resultados de la consulta en el servidor de aplicaciones.

De forma similar, para los formularios de introducción de fechas y horas o para las entradas que requieren validación de datos es preferible generarlos llamando a funciones apropiadas. Estas funciones pueden generar códigos en JavaScript que realizan la validación en el navegador.

Una tarea habitual de muchas aplicaciones de base de datos es la presentación de un conjunto de resultados. Es posible construir una función genérica que a partir de una consulta SQL (o un ResultSet), como argumento, muestre las tuplas del resultado de la consulta (o del ResultSet), en forma de tabla. Se pueden utilizar llamadas a metadatos de JDBC para disponer de información como el número de columnas y el nombre y tipos de las columnas del resultado de la consulta; esta información se utiliza para mostrar apropiadamente el resultado.

Para tratar las situaciones en las que el resultado de una consulta es muy grande, estas funciones para mostrar el resultado pueden realizar también la *paginación* del mismo. La función puede mostrar solo un cierto número de registros en una página y proporcionar controles para pasar a la página siguiente o anterior, o ir a una determinada página de resultados.

Por desgracia, no existe una API estándar de Java, que se use extensamente, para funciones del tipo indicado anteriormente. La construcción de este tipo de bibliotecas puede ser un proyecto interesante de programación.

Sin embargo, existen herramientas como el framework JavaServer Faces (JSF), que dispone de las características mencionadas anteriormente. El framework JSF incluye una biblioteca de etiquetas JSP que implementa estas características. El IDE Netbeans tiene un componente llamado VisualWeb que permite construir con JSF, mediante un entorno visual de desarrollo en el que los componentes de la interfaz de usuario se pueden arrastrar en la página y personalizar sus propiedades.

Por ejemplo, JSF tiene componentes para crear menús desplegables, o mostrar una tabla que se puede configurar para obtener los datos de una consulta a una base de datos.

JSF también permite especificar la validación para componentes, por ejemplo, para que una selección o entrada sea obligatoria o para restringir los valores de una fecha dentro de un determinado intervalo.

Active Server Pages (ASP) de Microsoft, y su versión más reciente, Active Server Pages.NET(ASP.NET), es una alternativa a JSP/Java muy utilizada. ASP.NET es similar a JSP en que el código en un lenguaje como Visual Basic o C# se puede incorporar al código HTML. Además, ASP.NET proporciona muchos controles, que se interpretan en el servidor, y generan HTML que se envía al cliente. Estos controles pueden simplificar significativamente la construcción de interfaces web. Posteriormente se dará una descripción general de las ventajas de estos controles.

Por ejemplo, los controles como los menús desplegables y las listas se pueden asociar a un objeto DataSet. Los objetos DataSet son similares a los objetos ResultSet de JDBC, y se suelen crear para ejecutar una consulta a una base de datos. El contenido del menú en HTML se genera con el contenido del objeto DataSet; por ejemplo, una consulta puede obtener el nombre de todos los departamentos de una organización en un DataSet y el menú asociado debería contener esos nombres. Por tanto, el menú que dependa del contenido de la base de datos se puede crear de esta manera de forma muy conveniente con muy poca programación.

Se pueden añadir controles de validación a los campos de entrada del formulario; mediante una especificación declarativa se pueden restringir intervalos de valores, o si la entrada es obligatoria y el usuario tiene que introducir un valor. El servidor crea el código HTML apropiado combinado con JavaScript para realizar la validación en el navegador del usuario. Los mensajes de error que se muestran para cada entrada no válida se asocian en cada control de validación.

Se puede especificar que las acciones de usuario generen una acción asociada en el servidor. Por ejemplo, se puede especificar que cuando se seleccione una opción de menú genere una acción asociada en el lado del servidor (se genera código JavaScript para

detectar el evento de selección e iniciar la acción en el lado del servidor). El código en VisualBasic/C# que muestra los datos pertenecientes al valor seleccionado se pueden asociar con la acción en el lado del servidor. Por tanto, seleccionar una opción de un menú puede hacer que se actualicen los datos que se muestran en la página sin necesidad de que el usuario haga clic en un botón de enviar.

El control DataGrid proporciona un mecanismo muy apropiado para mostrar los resultados de una consulta. Un DataGrid se asocia con un objeto DataSet, que suele ser el resultado de una consulta. El servidor genera el código HTML que muestra el resultado de la consulta en forma de tabla. Las cabeceras de las columnas se generan automáticamente a partir de los metadatos del resultado. Además, los DataGrid proporcionan funciones como la paginación y permiten al usuario ordenar los resultados por columnas. Todo el código HTML, así como la funcionalidad en el lado del servidor para implementar estas funciones, se genera automáticamente en el servidor. El DataGrid también permite que el usuario pueda editar los datos y enviar los cambios de vuelta al servidor. El desarrollador puede especificar una función que se ejecuta cuando se edita una fila, lo que permite actualizar la base de datos.

Microsoft Visual Studio proporciona una interfaz gráfica de usuario para la creación de estas funciones, reduciendo en gran medida el esfuerzo de programación.

Véanse las notas bibliográficas para más información sobre ASP.NET.

9.5.2. Frameworks para aplicaciones web

Existen varios frameworks de desarrollo de aplicaciones web que proporcionan funciones muy utilizadas como:

- Un modelo orientado a objetos con una correspondencia entre objetos y relaciones para almacenar los datos en una base de datos relacional, como ya se vio en la Sección 9.4.2.
- Un mecanismo, relativamente declarativo, para especificar un formulario con restricciones de validación para las entradas del usuario, a partir del cual el sistema genera código HTML y JavaScript/Ajax para implementar el formulario.
- Un sistema de plantillas de guiones (similar a JSP).
- Un controlador que hace corresponder los eventos de interacción de usuario, como los envíos de formularios, con las funciones apropiadas para manejar los eventos. El controlador también maneja la autenticación y las sesiones. Algunos frameworks también proporcionan herramientas para la gestión de autorizaciones.

Estos frameworks proporcionan diversas características necesarias para la construcción de aplicaciones web de forma integrada. Al generar formularios a partir de especificaciones declarativas, y manejar el acceso a los datos de forma transparente, el framework minimiza la cantidad de código que el programador tiene que escribir para realizar la aplicación web.

Existe un gran número de este tipo de frameworks, para distintos lenguajes. Algunos de los más utilizados incluyen Ruby on Rails, basado en el lenguaje de programación Ruby, JBoss Seam, Apache Struts, Swing, Tapestry y WebObjects, todos ellos basados en Java/JSP. Algunos de ellos, como Ruby on Rails y JBoss Seam proporcionan una herramienta para crear automáticamente interfaces de usuario **CRUD** simples; es decir, interfaces que permiten crear, leer, actualizar y borrar objetos/tuplas, generando código a partir de un modelo de objetos o de una base de datos. Estas herramientas son particularmente útiles para empezar a generar aplicaciones simples de forma muy rápida, y el código generado se puede editar para construir interfaces web más sofisticadas.

9.5.3. Generadores de informes

Los generadores de informes son herramientas para generar informes legibles a partir de las bases de datos. Integran la consulta a la base de datos con la creación de texto con formato y gráficos-resumen (como los gráficos de barras o de tarta). Por ejemplo, un informe puede mostrar las ventas totales de los últimos dos meses para cada región de ventas.

El desarrollador de aplicaciones puede especificar el formato de los informes mediante los servicios de formato del generador de informes. Se pueden usar variables para almacenar parámetros como el mes y el año, así como para definir los campos del informe. Las tablas, gráficos de líneas, gráficos de barras u otros gráficos se pueden definir mediante consultas a la base de datos. Las definiciones de las consultas pueden hacer uso de los valores de los parámetros almacenados en las variables.

Una vez definida la estructura de los informes en un servicio generador de informes, se puede almacenar y ejecutar en cualquier momento para generar informes. Los sistemas generadores de informes ofrecen gran variedad de servicios para estructurar el resultado tabular, como la definición de encabezados de tablas y de columnas, la visualización de los subtotales correspondientes a cada grupo de la tabla, la división automática de las tablas de gran tamaño en varias páginas y la visualización de los subtotales al final de cada página.

La Figura 9.12 es un ejemplo de informe con formato. Los datos del informe se generan mediante la agregación de la información sobre los pedidos.

Gran variedad de marcas, como Crystal Reports y Microsoft (SQL Server Reporting Services), ofrecen herramientas para la generación de informes. Varias familias de aplicaciones, como Microsoft Office, incluyen procedimientos para incrustar directamente en los documentos los resultados con formato de las consultas a la base de datos. Los servicios de generación de gráficos proporcionados por Crystal Reports, o por hojas de cálculo como Excel se pueden usar para tener acceso a las bases de datos y generar descripciones tabulares o gráficas de los datos. En los documentos de texto creados con Microsoft Word, por ejemplo, se puede incrustar uno o más de estos gráficos. Los gráficos se crean inicialmente a partir de datos generados ejecutando consultas a la base de datos; las consultas se pueden volver a ejecutar y los gráficos se pueden regenerar cuando sea necesario, para crear una versión actual del informe global.

Además de generar los informes estáticos, estas herramientas permiten que sean interactivos. Así, un usuario puede «profundizar» en áreas de su interés, por ejemplo, pasar de una vista agregada que muestre las ventas totales por año completo a las cifras de ventas mensuales en un año concreto. El análisis interactivo de datos ya se trató anteriormente en la Sección 5.6.

Acme Supply Company, Inc.
Informe de ventas trimestrales

Periodo: 1 de enero a 31 de marzo, 2012

Región	Categoría	Ventas	Subtotal
Norte	Hardware de computadora	1.000.000	
	Software de computadora	500.000	
	Todas las categorías		1.500.000
Sur	Hardware de computadora	200.000	
	Software de computadora	400.000	
	Todas las categorías		600.000
		Ventas totales	2.100.000

Figura 9.12. Un informe con formato.

9.6. Rendimiento de la aplicación

Puede que millones de personas de todo el mundo accedan a un determinado sitio web, a tasas de miles de peticiones por segundo o incluso mayores, en los sitios más populares. Asegurar que las peticiones se sirven con un tiempo de respuesta bajo es un gran desafío para los desarrolladores web. Para ello, los desarrolladores intentan acelerar el procesamiento de cada petición usando técnicas como el caché, el procesamiento paralelo o utilizando múltiples servidores web. A continuación se describen brevemente estas técnicas. El ajuste de las aplicaciones de bases de datos se describe con más detalle más adelante en el Capítulo 24 (Sección 24.1).

9.6.1. Reducir la sobrecarga mediante cachés

Se utilizan distintos tipos de técnicas de caché para aprovechar las partes comunes de diversas transacciones. Por ejemplo, suponga el código de la aplicación para servir las necesidades de cada petición de conectarse a la base de datos con JDBC. La creación de una nueva conexión JDBC puede tardar varios milisegundos, por lo que abrir una nueva conexión para cada petición de un usuario no es una buena idea si las tasas de transacciones son muy altas.

El método de **bancos de conexiones** se usa para reducir esta sobrecarga; funciona de la siguiente forma: el gestor del banco de conexiones, una parte del servidor de aplicaciones, crea un banco, es decir, un conjunto de conexiones ODBC/JDBC abiertas. En lugar de abrir una nueva conexión a la base de datos, el código que da el servicio a una petición de usuario, normalmente un servlet, solicita una conexión del banco de conexiones y devuelve la conexión al banco cuando el código, del servlet, termina su procesamiento. Si el banco no dispone de conexiones sin usar cuando se solicita, se abre una nueva conexión a la base de datos, con cuidado de no exceder el número máximo de conexiones que el sistema de la base de datos admite concurrentemente. Si durante un periodo de tiempo hay muchas conexiones abiertas sin usarse, el gestor del banco de conexiones puede cerrar algunas de ellas. Muchos servidores de aplicaciones y los nuevos controladores de ODBC/JDBC proporcionan un gestor de bancos de conexiones.

Un error común que cometen muchos programadores cuando crean aplicaciones web es olvidarse de cerrar una conexión JDBC abierta, o de forma equivalente cuando se usa un banco de conexiones, olvidarse de devolver la conexión al banco. Cada petición abre una nueva conexión a la base de datos, y la base de datos llega al límite del número de conexiones que puede tener abiertas a la vez. Estos problemas no aparecen cuando se prueba a pequeña escala, ya que las bases de datos suelen permitir cientos de conexiones abiertas, pues aparecen solo durante un uso intensivo. Algunos programadores asumen que las conexiones, como la memoria que se pide en los programas Java, se recolectan automáticamente. Desafortunadamente, no es así, y los programadores son los responsables de cerrar las conexiones que han abierto.

Ciertas peticiones pueden crear una consulta que ya se había enviado a la base de datos. El coste de comunicarse con la base de datos se puede reducir de forma importante haciendo un caché de los resultados de peticiones anteriores y reutilizándolos, mientras los resultados de la consulta no cambien en la base de datos. Algunos servidores web admiten este tipo de caché de resultados, sino se puede realizar explícitamente en el código de la aplicación.

El coste se puede reducir aún más haciendo caché de la página web final que se envía como respuesta a una petición. Si una nueva petición viene con los mismos parámetros que una previa, la solicitud no realiza ninguna actualización y la página web resultante está en la caché, por lo que se puede reutilizar y evitar el coste de volver a recalcular la página. El caché se puede utilizar en el nivel de fragmentos de páginas web, que se ensamblan para crear páginas web completas.

El caché de resultados de consultas y el de páginas web son formas de vistas materializadas. Si los datos de la base de datos subyacente cambian, los resultados en caché deben ser descartados o recalculados, o actualizados incrementalmente como en el mantenimiento de vistas materializadas (se describe más adelante en la Sección 13.5). Algunos sistemas de bases de datos, como Microsoft SQL Server, proporcionan una forma para que el servidor de aplicaciones registre una consulta en la base de datos y obtenga una **notificación** de la base de datos cuando el resultado de la consulta cambie. Este mecanismo de notificaciones se puede usar para garantizar que los resultados de una consulta en caché en el servidor de aplicaciones siempre están actualizados.

9.6.2. Procesamiento paralelo

Una forma habitual de manejar una gran carga es utilizar un gran número de servidores de aplicación ejecutándose en paralelo, cada uno de ellos manejando una fracción de las peticiones. Se puede usar un servidor web o un router de red para encaminar las peticiones de los clientes a los distintos servidores de aplicaciones. Todas las peticiones de una sesión de un determinado cliente deben ir al mismo servidor de aplicaciones, ya que el servidor mantiene el estado de una sesión del cliente. Esta propiedad se puede asegurar, por ejemplo, encaminando todas las peticiones de una determinada dirección de IP al mismo servidor de aplicaciones, de forma que los usuarios vean una vista consistente de la base de datos.

Con la arquitectura anterior, la base de datos se podría convertir en un cuello de botella, al ser compartida. Los diseñadores de aplicaciones ponen una atención especial en minimizar el número de peticiones a la base de datos mediante el caché de consultas en el servidor de aplicaciones, como ya se ha tratado. Los sistemas de bases de datos paralelos se describen en el Capítulo 18.

9.7. Seguridad de las aplicaciones

La seguridad de las aplicaciones tiene que tratar con distintos frentes de seguridad y temas que van más allá de los que manejan las autorizaciones de SQL.

El primer punto en el que reforzar la seguridad es en la aplicación. Para ello, las aplicaciones deben autenticar a los usuarios y asegurar que solo se les permite realizar las tareas autorizadas.

Hay muchas formas en que se puede ver comprometida la seguridad de una aplicación, incluso aunque el sistema de base de datos sea seguro, debido a una aplicación mal escrita. En esta sección, en primer lugar se describen algunos agujeros de seguridad que pueden permitir que los hackers lleven a cabo acciones saltándose las comprobaciones de autenticación y autorización de la aplicación, y se explica cómo evitar estos agujeros de seguridad. Más adelante en esta sección se describen técnicas para la autenticación segura y para las autorizaciones de grano fino. Posteriormente se describen las trazas de auditoría que pueden ayudar a recuperarse de un acceso no autorizado y de actualizaciones erróneas. La sección concluye describiendo algunos elementos de privacidad de los datos.

9.7.1. Inyección SQL

En los ataques de **inyección SQL,** el atacante consigue que una aplicación ejecute una consulta SQL creada por el atacante. En la Sección 5.1.1.4 se vio un ejemplo de vulnerabilidad de inyección SQL si las entradas del usuario se concatenaban directamente con una consulta SQL y se enviaban a la base de datos. Como ejemplo

adicional de inyección SQL, considere el formulario que se muestra en la Figura 9.4. Suponga que el servlet que se muestra en la Figura 9.8 crea una cadena de consulta en SQL que usa la siguiente expresión en Java:

```
String consulta = «select * from estudiante where nombre like '%"
                + nombre + "%'»
```

donde nombre es una variable que contiene la entrada del usuario, y se ejecuta la consulta en la base de datos. Un atacante malicioso podría usar el formulario web y escribir una cadena de texto como «';<una sentencia de SQL >; −− », donde <una sentencia de SQL> indica cualquier sentencia de SQL que el atacante desee, en lugar de un nombre válido de estudiante. El servlet ejecutaría entonces la siguiente consulta:

select * **from** *estudiante* **where** *nombre* **like** ' ';
<una sentencia de SQL >;−−'

La comilla insertada por el atacante cierra la cadena, el siguiente punto y coma termina la consulta y el texto que hay a continuación, que inserta el atacante, se interpreta como una segunda consulta de SQL, y la comilla de cierre se ha comentado. Por tanto, el malicioso usuario ha conseguido insertar una sentencia de SQL arbitraria que ejecuta la aplicación. La sentencia puede causar un daño importante, ya que puede realizar cualquier acción en la base de datos, saltándose todas las medidas de seguridad implementadas en el código.

Como ya se trató en la Sección 5.1.1.4, para evitar estos ataques es mejor utilizar sentencias ya preparadas para ejecutar las consultas de SQL. Cuando se configura un parámetro en una consulta preparada, JDBC añade automáticamente caracteres de escape de forma que las comillas que incluya el usuario ya no permiten terminar una cadena. De forma similar, se podría aplicar una función que añada estos caracteres de escape a las cadenas de entrada antes de que se concatenen con las consultas SQL, en lugar de usar sentencias preparadas.

Otra fuente de riesgo de inyección SQL proviene de las aplicaciones que crean las consultas de forma dinámica a partir de condiciones de selección y ordenación de los atributos especificados en un formulario. Por ejemplo, una aplicación permite al usuario especificar qué atributos se deberían utilizar para ordenar los resultados de una consulta. Se construye la consulta apropiada en SQL de acuerdo con los atributos especificados. Suponga que la aplicación recoge el nombre del atributo del formulario, en la variable ordenAtributo y crea una cadena de consulta de la forma:

```
String consulta = «select * from takes order by» + ordenAtributo;
```

Un usuario malicioso puede enviar una cadena arbitraria en lugar de un valor de ordenAtributo con sentido, incluso aunque se use el formulario HTML para obtener la entrada y restringir los valores que proporcione el menú. Para evitar este tipo de inyección SQL, la aplicación debería asegurarse de que el valor de la variable ordenAtributo es uno de los valores permitidos (en el ejemplo, los nombres de los atributos) antes de concatenarlo.

9.7.2. Secuencias de comandos en sitios cruzados y falsificación de peticiones

Un sitio web que permite a los usuarios introducir texto, como un comentario o un nombre, y después lo almacena para mostrarlo más tarde a otros usuarios es potencialmente vulnerable a un tipo de ataque llamado **secuencias de comandos en sitios cruzados** (XXS: *cross-site scripting*). En estos ataques, un usuario malicioso introduce código escrito en el lenguaje de secuencias de comandos como JavaScript o Flash, en lugar de un nombre válido o un comentario. Cuando un usuario diferente ve el texto introducido, el navegador puede ejecutar una secuencia de comandos y llevar a cabo acciones como el envío de información de cookies privadas de vuelta al usuario malicioso o, incluso, ejecutar una acción en un servidor web diferente a aquel en el que el usuario se había registrado.

Por ejemplo, suponga que el usuario consigue registrarse en su cuenta del banco cuando se ejecuta la secuencia de comandos. Podría enviar la información de la cookie relacionada con el registro en la cuenta del banco de vuelta al usuario malicioso, quien podría usar la información para conectarse al servidor web del banco, engañándolo y haciéndole creer que es una conexión del usuario original. O, el guion podría acceder a las páginas apropiadas del sitio web del banco, con el conjunto de parámetros apropiados, y ejecutar transferencias de dinero. De hecho este problema en particular puede ocurrir incluso sin secuencias de comandos con una simple línea de código como:

```
<img src=
  "http://mybank.com/transfermoney?amount=
   1000&toaccount=14523">
```

suponiendo que la URL mybank.com/transfermoney acepta los parámetros especificados y lleva a cabo la transferencia de dinero. Esta última vulnerabilidad también se llama **falsificación de petición en sitios cruzados** o **XSRF** (también denominada **CSRF**).

XSS se puede realizar de otras formas como tentando al usuario a visitar un sitio web con secuencias de comandos maliciosas en sus páginas. Existen otros tipos de ataques XSS y XSRF más complejos, que no se van a tratar. Para protegerse contra estos ataques se deben tener en cuenta dos recomendaciones:

- **Evitar que un sitio web se use para lanzar ataques XSS o XSRF.** La forma más sencilla es impedir que los usuarios puedan escribir cualquier tipo de etiqueta HTML. Existen funciones para detectar o eliminar tales etiquetas. Estas funciones se pueden utilizar para evitar etiquetas HTML y por tanto impedir que cualquier secuencia de comandos se muestre al usuario. En algunos casos el formato de HTML es útil y en tal caso las funciones que analizan el texto permiten un número limitado de etiquetas impidiendo otras que pueden ser peligrosas; hay que diseñarlo cuidadosamente, ya que algo tan inocente como un programa para mostrar imágenes puede explotarse maliciosamente.
- **Proteger el sitio web de ataques XSS o XSRF que se lanzan desde otros sitios.** Si el usuario se ha registrado en un sitio web y visita otro diferente vulnerable a XSS, el código malicioso que se ejecuta en el navegador del usuario podría ejecutar acciones en tu sitio web, o pasar información de sesión relacionada con tu sitio web de vuelva al usuario malicioso que tratará de explotarla. Esto no se puede prevenir, pero se pueden tomar algunas medidas para minimizar los riesgos.
  - El protocolo HTTP permite que un servidor compruebe el **referente** de un acceso de una página, es decir, la URL de la página que tiene el enlace en el que el usuario ha hecho clic para iniciar el acceso a la página. Comprobando que el referente es válido, por ejemplo, que tiene una URL del mismo sitio web, se pueden prevenir los ataques XSS originados desde una página web diferente a la que accede el usuario.
  - En lugar de utilizar solo las cookies para identificar una sesión, también se puede restringir la dirección IP desde la que se realizó la autenticación originalmente. Con ello, aunque un usuario malicioso consiga una cookie, no podría utilizarla para acceder desde una computadora diferente.
  - No utilice nunca el método GET para realizar las actualizaciones. Esto evita los ataques que utilizan <img src…>, como el que se vio anteriormente. De hecho, la norma HTTP recomienda que los métodos GET nunca realicen actualizaciones, por otras razones como que el refresco de una página repite una acción que solo debería producirse una vez.

9.7.3. Fuga de contraseñas

Otro problema que deben tratar los desarrolladores de aplicaciones es el almacenamiento de las contraseñas en texto claro en el código de la aplicación. Por ejemplo, programas de secuencia de guiones como JSP suelen incluir contraseñas en texto claro. Si estos programas se guardan en un directorio accesible al servidor web, un usuario externo puede ser capaz de acceder al código fuente del programa y conseguir acceso a la contraseña de la cuenta de la base de datos que usa la aplicación. Para evitar estos problemas, muchos servidores de aplicación proporcionan mecanismos para guardar las contraseñas de forma cifrada, que el servidor descifra antes de pasarlas a la base de datos. Esta característica elimina la necesidad de almacenar las contraseñas en texto claro en los programas de aplicación. Sin embargo, si la clave de cifrado sigue siendo vulnerable, este mecanismo puede no ser muy efectivo.

Otra medida para no comprometer la contraseña de la base de datos es que muchos sistemas de base de datos permiten que el acceso a esta se restrinja a un intervalo de direcciones IP, normalmente las máquinas en las que se ejecutan los servidores de aplicaciones. El intento de conexión a la base de datos desde otras direcciones de Internet se rechaza. Por tanto, a no ser que el usuario malicioso sea capaz de registrarse en el servidor de aplicaciones, no podrá hacer daño, incluso aunque consiga acceso a la contraseña de la base de datos.

9.7.4. Autenticación de las aplicaciones

La autenticación se refiere a la tarea de verificar la identidad de una persona/software que se conecta a una aplicación. La forma más simple de autenticación consiste en el uso de una contraseña secreta que hay que presentar cuando un usuario se conecta a la aplicación. Desafortunadamente las contraseñas se comprometen fácilmente, por ejemplo, adivinándolas o leyendo los paquetes de la red si estas no se envían cifradas. Para aplicaciones críticas, como los entornos de banca, se necesitan esquemas más robustos. El cifrado es la base de los esquemas de autenticación robustos. La autenticación mediante cifrado se trata en la Sección 9.8.3.

Muchas aplicaciones utilizan la **autenticación en dos pasos,** en la que existen dos *pasos* distintos, es decir, piezas de información o procesos que se usan para identificar al usuario. Estos dos factores no deberían compartir una vulnerabilidad común; por ejemplo, si un sistema requiere dos contraseñas, ambas podrían comprometerse en una fuga de contraseñas de la misma forma; mediante análisis del tráfico de red o mediante un virus en la computadora del usuario. Aunque se pueden usar medidas biométricas como la huella dactilar o los escáneres de iris, no son habituales en la red.

Las contraseñas se usan como primer paso en la mayoría de los esquemas de autenticación en dos pasos. Como segundo paso ampliamente usado, se suelen usar tarjetas inteligentes u otros dispositivos cifrados conectados mediante una interfaz USB (véase la Sección 9.8.3).

Los dispositivos de contraseñas de un solo uso, que generan un nuevo número pseudoaleatorio, por ejemplo cada minuto, también son muy utilizados como segundo paso. Cada usuario dispone de uno de estos dispositivos y para identificarse debe introducir el número que muestra el dispositivo cuando realiza la autenticación, junto con la contraseña. Cada dispositivo genera una secuencia diferente de números pseudoaleatorios. El servidor de aplicaciones puede generar la misma secuencia de números pseudoaleatorios que el dispositivo del usuario, parando cuando llegue al número que se muestra durante el proceso de autenticación y verificando que los números coinciden. Este esquema requiere que el reloj del dispositivo y el del servidor estén razonablemente sincronizados.

Otro segundo paso es el envío de un SMS con un número aleatoriamente generado como contraseña de un solo uso al teléfono del usuario, cuyo número se registró previamente, siempre que el usuario desee registrarse en la aplicación. El usuario tiene que tener un teléfono con dicho número en el que recibir el SMS y, después, introducir la contraseña de un solo uso junto con su contraseña habitual, para autenticarse.

Hay que tener en cuenta que incluso en la autenticación en dos pasos, los usuarios siguen siendo vulnerables a un ataque de **hombre en el medio.** En estos ataques, un usuario que intenta conectarse a la aplicación es desviado a un sitio web falso, que acepta la contraseña, incluyendo la contraseña del segundo paso, y la usa inmediatamente para autenticarse en la aplicación original. El protocolo HTTPS, que se describe más adelante en la Sección 9.8.3.2, se usa para autenticar a los usuarios de un sitio web de forma que el usuario no se conecte a un sitio falso creyendo que era el sitio que pretendía. El protocolo HTTPS cifra los datos y, por tanto, evita el ataque de hombre en el medio.

Cuando los usuarios acceden a varios sitios web, suele resultar tedioso que el usuario tenga que autenticarse en cada uno por separado, normalmente con contraseñas diferentes en cada uno de ellos. Existen sistemas que permiten que el usuario se autentique en un servicio central de autenticación; la misma contraseña sirve para acceder a múltiples sitios web. El protocolo LDAP se usa extensamente para implementar tal punto central de autenticación; las organizaciones implantan un servidor de LDAP con la información de los nombres y contraseñas de los usuarios y las aplicaciones usan el servidor de LDAP para la autenticación de los usuarios.

Además de para la autenticación de usuarios, el servicio central de autenticación se puede utilizar, por ejemplo, para proporcionar información centralizada sobre el usuario, como su nombre, correo electrónico, dirección, etc. a la aplicación. Esto evita introducir esta información en cada aplicación. LDAP se puede usar para esta función, como se describe más adelante en la Sección 19.10.2. Otros sistemas de directorio, como los directorios activos de Microsoft, también proporcionan mecanismos para la autenticación de usuarios, así como para facilitar información de los mismos.

Un sistema de **acceso con firma única** (*single sign-on*) permite que los usuarios se autentiquen una sola vez y distintas aplicaciones puedan verificar la identidad del usuario mediante un servicio de autenticación sin requerir que se vuelvan a autenticar. En otras palabras, una vez que el usuario se ha registrado en un sitio, no tiene que volver a introducir su nombre de usuario y contraseña en el resto de sitios que utilizan el mismo servicio de acceso con firma única. Estos mecanismos de acceso con firma única se han usado en protocolos de autenticación de redes como Kerberos, y existen implementaciones disponibles para su uso en aplicaciones web.

El **lenguaje de marcado para confirmaciones de seguridad** (SAML: *Security Assertion Markup Language*) es una norma para el intercambio de información de autenticación y autorización entre distintos dominios de seguridad, que proporciona acceso de firma única entre organizaciones. Por ejemplo, suponga que una aplicación necesita proporcionar acceso a todos los estudiantes desde una determinada universidad, digamos Yale. La universidad puede configurar un servicio basado en web que lleve a cabo la autenticación. Suponga que un usuario se conecta a la aplicación con un nombre de usuario como «joe@yale.edu». La aplicación, en lugar de autenticar al usuario, desvía a este al servicio de autenticación de la Universidad de Yale, que es quien autentica al mismo y, después, dice a la aplicación quién es el usuario y puede indicarle información adicional como la categoría del mismo (estudiante o profesor) u otra información relevante. La contraseña del usuario y otros pasos de autenticación nunca se revelan a la aplicación, y el usuario no necesita registrarse de forma explícita con la misma. Sin embargo, la aplicación debe confiar en el servicio de autenticación de la universidad cuando autentica a un usuario.

La norma **OpenID** es una forma alternativa de acceso con firma única entre organizaciones y ha incrementado su aceptación en los últimos años. Un gran número de sitios web populares como Google, Microsoft, Yahoo! y otros, actúan como proveedores de autenticación OpenID. Cualquier aplicación que actúe como cliente de OpenID puede utilizar estos proveedores para autenticar a un usuario; por ejemplo, un usuario que tiene una cuanta en Yahoo! puede elegir Yahoo! como proveedor de autenticación. El usuario se ve desviado a Yahoo! para la autenticación y si se efectúa correctamente se le redirige de forma transparente de nuevo a la aplicación y puede continuar utilizándola.

9.7.5. Autorización al nivel de aplicación

Aunque la norma de SQL admite un sistema bastante flexible de autorización basado en roles, descritos en la Sección 4.6, el modelo de autorización de SQL desempeña un papel muy limitado en la gestión de las autorizaciones de usuario en una aplicación típica. Por ejemplo, suponga que desea que todos los estudiantes vean sus notas, pero no las notas de los otros. Esta autorización no se puede especificar en SQL por las siguientes dos razones:

1. **Falta de información sobre el usuario final.** Con el crecimiento de la web, el acceso a las bases de datos proviene principalmente de servidores de aplicaciones web. Los usuarios finales normalmente no tienen identificadores de usuario individuales en la propia base de datos y, por tanto, puede que solo haya un único identificador de usuario en la base de datos para todos los usuarios de un servidor de aplicaciones. Por tanto, en el escenario anterior no se puede usar la especificación de autorización en SQL.

   Es posible que un servidor de aplicaciones autentique usuarios finales y, entonces, traslade la información de autenticación a la base de datos. En esta sección se supondrá que la función *syscontext.user id*() devuelve el identificador del usuario de la aplicación en cuyo nombre se ejecuta la consulta.[6]
2. **Falta de autorización de grano fino.** La autorización debe realizarse en el nivel de las tuplas individuales, si se desea autorizar a los estudiantes a que vean solo sus propias notas. Esta autorización no es posible en norma SQL actual, que solo permite la autorización sobre una relación completa o una vista o sobre determinados atributos de relaciones o vistas.

   Se puede intentar solventar esta limitación creando para cada estudiante una vista de la relación *matricula* que únicamente muestre las notas de dicho estudiante. Aunque en principio podría funcionar, sería muy costoso, ya que se debería crear una vista para cada uno de los estudiantes matriculados en la universidad, lo que resulta poco práctico.[7]

   Una alternativa es crear una vista de la forma:

   create view *estudianteMatricula* **as**

   select *

   from *matricula*

   where *matricula.ID* = *syscontext.user_id*()

   Los usuarios obtienen autorización para esta vista, en lugar de para toda la relación *matricula* subyacente. Sin embargo, las consultas que se ejecutan en lugar del estudiante deben escribirse en la vista *estudianteMatricula*, en lugar de en la relación *matricula* original; mientras que las consultas ejecutadas en lugar de los profesores puede que necesiten una vista diferente. El resultado final es que la tarea de desarrollar la aplicación se vuelve más compleja.

La tarea de autenticación normalmente se lleva a cabo enteramente en la aplicación, sin las facilidades de autorización de SQL. En el nivel de aplicación, los usuarios consiguen la autorización para determinadas interfaces de acceso, y se les puede restringir la vista o actualización de ciertos datos.

Al realizar la autorización en la aplicación se consigue un alto grado de flexibilidad para los desarrolladores de aplicación, pero presenta ciertos problemas:

- El código de comprobación de la autorización se mezcla con el resto del código de la aplicación.
- La implementación de la autorización en el código de la aplicación, en lugar de especificarlo de forma declarativa en SQL, hace que sea más difícil asegurar la ausencia de agujeros de seguridad. Como resultado, puede que alguno de los programas de aplicación no compruebe la autorización y permita que usuarios no autorizados tengan acceso a datos confidenciales.

Verificar que todos los programas de aplicación realizan las comprobaciones de autorización necesarias implica revisar todo el código del servidor de aplicaciones, una tarea formidable para un gran sistema. En otras palabras, las aplicaciones tienen una gran superficie expuesta, lo que hace que la tarea de proteger a la aplicación sea mucho más difícil. De hecho, se han encontrado agujeros de seguridad en muchas aplicaciones reales.

En contraste, si una base de datos dispone directamente de autorizaciones de grano fino, las políticas de autorización se pueden especificar y forzar en el nivel de SQL, que tiene una superficie expuesta mucho más pequeña. Incluso, aunque algunas interfaces de la aplicación omitan inadvertidamente las comprobaciones de autorización necesarias, el nivel de autorización de SQL evitaría que se ejecutasen acciones no autorizadas.

Algunos sistemas de base de datos proporcionan mecanismos de autorización de grano fino. Por ejemplo, **la base de datos privada virtual** (VPD: *Virtual Private Database*) de Oracle permite que un administrador del sistema asocie una función con una relación; esta función devuelve un predicado que se añade a todas las consultas que utilicen la relación (se pueden añadir diferentes funciones para las actualizaciones). Por ejemplo, usando nuestra sintaxis para obtener los identificadores de usuario de la aplicación, la función de la relación *matricula* puede devolver un predicado como:

$$ID = sys\ context.user_id()$$

Este predicado se añade a la cláusula **where** de todas las consultas que usen la relación *matricula*. El resultado, suponiendo que el programa de aplicación configura el valor de *user_id* al *ID* del estudiante, es que cada estudiante solo verá las tuplas correspondientes a los cursos en que se haya matriculado.

Por tanto, VPD proporciona autorización en el nivel de determinadas tuplas o filas de una relación, y se puede decir, en este sentido, que es un mecanismo de *autorización en el nivel de fila*. Un problema potencial de añadir un predicado como el descrito es que puede cambiar de forma importante el significado de una consulta. Por ejemplo, si un usuario escribe una consulta para encontrar la nota media de todas las asignaturas, debería obtener el promedio de *sus* notas, no de todas las notas. Aunque el sistema le diese la respuesta «correcta» para la consulta reescrita, la respuesta no se correspondería con la consulta que el usuario puede que desease realizar al enviarla.

Véanse las notas bibliográficas para más información sobre la VPD de Oracle.

6 En Oracle, una conexión de JDBC usando el controlador de JDBC de Oracle puede establecer el identificador de usuario final usando el método OracleConnection.setClientIdentifier(userId), y la consulta de SQL puede usar la función sys_context('USERENV', 'CLIENT_IDENTIFIER') para obtener el identificador de usuario.

7 Los sistemas de base de datos se diseñan para gestionar grandes relaciones, pero gestionan la información de esquema, como las vistas, de una forma que supone volúmenes de datos menores para mejorar el rendimiento total.

9.7.6. Trazas de auditoría

Una **traza de auditoría** es un registro de todos los cambios, inserciones, borrados y actualizaciones en los datos de la aplicación, junto con la información sobre el usuario que realizó los cambios y cuándo se hicieron. Si se rompe la seguridad de la aplicación o, aunque no se rompa la seguridad, se realiza alguna actualización de forma errónea, la traza de auditoría puede (a) ayudar a descubrir qué ha pasado y quién llevó a cabo las acciones, y (b) ayudar a reparar el daño realizado tras la brecha de seguridad o la actualización errónea.

Por ejemplo, si se descubre que una nota de un estudiante es incorrecta, se puede examinar el registro de auditoría para descubrir cuándo y cómo se modificó la nota, así como para descubrir quién realizó el cambio. La universidad también puede usar la traza de auditoría para trazar todas las actualizaciones realizadas por un usuario y descubrir otras actualizaciones incorrectas o fraudulentas y corregirlas.

Las trazas de auditoría también se pueden utilizar para detectar brechas de seguridad cuando una cuenta de usuario se ha visto comprometida y ha accedido con ella un intruso. Por ejemplo, cada vez que se registra un usuario se le puede informar de todas las actualizaciones en la traza de auditoría que se realizaron en el pasado reciente; si el usuario observa una actualización que no hizo, es probable que la cuenta esté comprometida.

Es posible crear trazas de auditoría definiendo los disparadores adecuados sobre las actualizaciones de las relaciones (usando variables definidas por el sistema que identifican el nombre de usuario y la hora). Sin embargo, muchos sistemas de bases de datos proporcionan mecanismos predeterminados para crear trazas de auditoría que son más cómodos de usar. Los detalles de la creación de las trazas de auditoría varían de unos sistemas de bases de datos a otros, y se deben consultar los manuales de los diferentes sistemas para conocer los detalles.

Las trazas de auditoría del nivel de base de datos suelen ser insuficientes para las aplicaciones, ya que no suelen ser capaces de trazar quién es el usuario final de la aplicación. Más aún, las actualizaciones se registran en un nivel bajo, en términos de la actualización de las tuplas de una relación, en lugar de en un nivel alto, en términos de la lógica de negocio. Por tanto, las aplicaciones normalmente crean una traza de auditoría de alto nivel, registrando, por ejemplo, qué acción se ha realizado, por quién, cuándo y desde qué dirección IP se originó la petición.

Un tema relacionado es el de la protección de la propia traza de auditoría de forma que no sea modificada o borrada por los usuarios que rompen una brecha de seguridad de la aplicación. Una solución consiste en realizar una copia de la traza de auditoría en una máquina diferente, a la que el intruso no pueda tener acceso, y que cada registro de la traza de auditoría se copie en cuanto se genere.

9.7.7. Privacidad

En un mundo en el que ha crecido la cantidad de datos personales accesibles online, las personas están cada vez más preocupadas por la privacidad de sus datos. Por ejemplo, la mayoría de las personas querrán que sus datos médicos se mantengan privados y no puedan hacerse públicos. Sin embargo, los datos médicos deben estar disponibles para los médicos y técnicos de emergencias que tratan al paciente. La mayoría de los países tienen leyes sobre la privacidad de estos datos que definen cuándo y quién puede acceder a ellos. El incumplimiento de las leyes sobre privacidad puede conllevar penas criminales en algunos países. Hay que construir con sumo cuidado las aplicaciones que acceden a estos datos privados, teniendo en cuenta las leyes sobre privacidad.

Por otra parte, los datos privados agregados desempeñan un importante papel en muchas tareas como la detección de efectos secundarios de los medicamentos o la detección de la propagación de una epidemia. Cómo hacer disponibles estos datos para los investigadores sin comprometer la privacidad de las personas es un problema real. Como ejemplo, suponga que un hospital oculta los nombres de los pacientes pero proporciona al investigador las fechas de nacimiento y los códigos postales (ambos útiles para el investigador). En muchos casos, con estos dos elementos de información se puede identificar de forma única a un paciente (con información de bases de datos externas), comprometiendo su privacidad. En esta situación en particular, una solución podría ser proporcionar el año de nacimiento pero no la fecha, junto con el código postal, lo que puede ser suficiente para el investigador. De esta forma no se proporciona información suficiente como para identificar de forma unívoca a la mayoría de las personas.[8]

Otro ejemplo: los sitios web suelen recopilar información de datos personales, como dirección, teléfono, dirección de email y tarjeta de crédito. Esta información puede ser necesaria para realizar una transacción de compra de un artículo en una tienda. Sin embargo, el cliente puede querer que esta información no esté disponible para otras organizaciones, o puede querer que cierta información, como el número de la tarjeta de crédito, se borre tras cierto periodo de tiempo como forma de evitar que caiga en manos no autorizadas en caso de que se produzca una brecha de seguridad. Muchos sitios web permiten a los clientes que indiquen sus preferencias de privacidad y deben asegurar que estas preferencias se respetan.

9.8. Cifrado y sus aplicaciones

El cifrado se refiere al proceso de transformar los datos de forma que no sean legibles, hasta que se aplica el proceso inverso de descifrado. Los algoritmos de cifrado usan una clave de cifrado para realizar el cifrado y necesitan una clave de descifrado (que puede ser la misma que la de cifrado o no, dependiendo del algoritmo utilizado), para realizar el descifrado.

El uso más antiguo del cifrado fue la transmisión de mensajes, cifrados usando una clave secreta que solo conocían el remitente y el receptor. Aunque el mensaje cayese en manos enemigas, el enemigo sin la clave no podía descifrar el mensaje ni entenderlo. El cifrado se usa extensamente en la actualidad para proteger los datos en tránsito en muy diversas aplicaciones, como la transferencia de datos en Internet y en las redes telefónicas móviles celulares. El cifrado también se usa para realizar otras tareas como la autenticación, como se ha visto en la Sección 9.8.3.

En el contexto de las bases de datos, el cifrado se usa para guardar los datos de forma segura, de modo que aunque se consigan por un usuario no autorizado, por ejemplo por el robo de un portátil que contiene los datos, no se pueda acceder a ellos sin la clave de descifrado.

Muchas bases de datos actuales guardan información sensible de clientes, como números de tarjetas de crédito, nombre, huellas dactilares, firmas y números de identificación; por ejemplo, en los Estados Unidos, el número de la seguridad social. Un criminal que acceda a estos datos puede usarlos de muchas formas en actividades ilegales, como la compra de bienes usando los números de las tarjetas de crédito o incluso solicitando una tarjeta de crédito en nombre de otra persona. Las organizaciones, como las compañías de tarjetas de crédito, utilizan el conocimiento de la información personal para identificar quién está solicitando un bien o un

8 En el caso de personas muy ancianas, que son relativamente infrecuentes, el año de nacimiento junto con el código postal pueden ser suficientes para identificarlas unívocamente, por lo que para las personas mayores de 80 años puede proporcionarse un intervalo de valores entre 80 años o más, en lugar de la edad real.

servicio. La filtración de esta información personal permite a un criminal suplantar a otra persona y acceder a bienes y servicios; esta suplantación se denomina **robo de identidad.** Por tanto, las aplicaciones que guardan este tipo de información sensible deben tener sumo cuidado de protegerla del robo.

Para reducir la posibilidad de que información sensible llegue a manos criminales, los países establecen por ley que cualquier base de datos que guarde información sensible debe guardarla de forma cifrada. Un negocio que no protege sus datos puede ser declarado culpable criminalmente en caso de robo. Por tanto, el cifrado es un componente crítico en cualquier aplicación que guarde información sensible.

9.8.1. Técnicas de cifrado

Existe un enorme número de técnicas para el cifrado de los datos. Puede que las técnicas de cifrado sencillas no proporcionen la seguridad adecuada, dado que para los usuarios no autorizados puede ser sencillo romper el código. Como ejemplo de técnica de cifrado débil considérese la sustitución de cada carácter por el siguiente en el alfabeto. Por tanto,

Perryridge

se transforma en:

Qfsszsjehf

Si un usuario no autorizado solo lee «Qfsszsjehf», probablemente no tenga la información suficiente para romper el código. Sin embargo, si el intruso ve gran número de nombres de sucursales cifrados, puede usar los datos estadísticos referentes a la frecuencia relativa de los caracteres para averiguar el tipo de sustitución realizado (por ejemplo, en español la *E* es la letra más frecuente, seguida por *A, O, L, S, M*, etc.).

Una buena técnica de cifrado posee las siguientes propiedades:

- Resulta relativamente sencillo para los usuarios autorizados cifrar y descifrar los datos.
- No depende del secreto del algoritmo, sino de un parámetro del algoritmo denominado *clave de cifrado*, que se usa para cifrar los datos. En una técnica de cifrado de **clave simétrica** la clave de cifrado también se usa para el descifrado. Al contrario, en una técnica de cifrado de **clave pública** (también conocida como **clave asimétrica**) existen dos claves diferentes, la clave pública y la clave privada, usadas para cifrar y descifrar los datos.
- Es extremadamente difícil para un intruso determinar la clave de cifrado, incluso aunque el intruso tenga acceso a datos cifrados. En el caso del cifrado con clave asimétrica, es extremadamente difícil inferir la calve privada, incluso aunque se disponga de la clave pública.

La **norma de cifrado avanzado** (AES: *Advanced Encryption Standard*) es un algoritmo de cifrado de clave simétrica que fue adoptado como norma de cifrado por el Gobierno de los EE.UU. en el año 2000, y ahora se usa extensamente. La norma se basa en el **algoritmo Rijndael** (por sus inventores V. Rijmen y J. Daemen). El algoritmo utiliza bloques de 128-bits de datos cada vez, mientras que la clave puede ser de 128, 192 o 256 bits de tamaño.

El algoritmo ejecuta un conjunto de pasos para desordenar los bits del bloque de datos, de forma que se pueda invertir el proceso durante el descifrado, y realiza una operación XOR con una «subclave» de 128 bits que se deriva de la clave de cifrado. Para cada uno de los bloques de datos que se cifra se genera una nueva subclave a partir de la clave de cifrado. Durante el descifrado, las subclaves se vuelven a generar a partir de la clave de cifrado y el proceso de cifrado se invierte para recuperar los datos originales. Anteriormente se usaba extensamente una norma llamada *norma de cifrado de datos* (DES: *Data Encryption Standard*), adoptada en 1977.

Para que cualquier esquema de cifrado de clave simétrica funcione, los usuarios deben obtener la clave de cifrado mediante un mecanismo seguro. Este requisito supone la mayor debilidad, ya que el esquema no es más seguro que la seguridad que ofrezca el mecanismo para transmitir la clave de cifrado.

El **cifrado de clave pública** es un esquema alternativo que evita parte de los problemas que se afrontan con las técnicas de cifrado de clave simétrica. Se basa en dos claves; una *clave pública* y otra *clave privada*. Cada usuario U_i tiene una clave pública E_i y una clave privada D_i. Todas las claves públicas están publicadas: cualquiera puede verlas. Cada clave privada solo la conoce el usuario al que pertenece. Si el usuario U_1 desea guardar datos cifrados, los cifra usando la clave pública E_1. Descifrarlos exige la clave privada D_1.

Como la clave de cifrado de cada usuario es pública, se puede intercambiar información de manera segura usando este esquema. Si el usuario U_1 desea compartir los datos con U_2, los codifica usando E_2, la clave pública de U_2. Dado que solo el usuario U_2 conoce la manera de descifrar los datos, la información se transmite de manera segura.

Para que el cifrado de clave pública funcione, debe haber un esquema de cifrado que haga extremadamente difícil de deducir la clave privada, dada la clave pública. Existe un esquema de ese tipo y se basa en estas condiciones:

- Que haya un algoritmo eficiente para comprobar si un número es primo o no.
- Que no se conozca ningún algoritmo eficiente para encontrar los factores primos de un número.

Para los fines de este esquema, los datos se tratan como conjuntos de enteros. Se crea la clave pública calculando el producto de dos números primos grandes: P_1 y P_2. La clave privada consiste en el par (P_1, P_2). El algoritmo de descifrado no se puede usar con éxito si solo se conoce el producto P_1P_2; necesita el valor de P_1 y de P_2. Dado que todo lo que se publica es el producto P_1P_2, los usuarios no autorizados necesitan poder factorizar P_1P_2 para robar los datos. Eligiendo P_1 y P_2 suficientemente grandes (por encima de 100 cifras) se puede hacer el coste de la factorización de P_1P_2 prohibitivamente elevado (del orden de años de tiempo de cálculo, incluso para las computadoras más rápidas).

En las notas bibliográficas se hace referencia a los detalles del cifrado de clave pública y a la justificación matemática de las propiedades de esta técnica.

Pese a que el cifrado de clave pública que usa el esquema descrito es seguro, también es costoso en cuanto a cálculo. Un esquema híbrido usado para proteger las comunicaciones es el siguiente: se genera de forma aleatoria una clave de cifrado simétrica (basada, por ejemplo, en AES) y se intercambia de forma segura usando un esquema de cifrado de clave pública, entonces se usa el cifrado de clave simétrica usando dicha clave para los datos que se transmitan después.

El cifrado de valores pequeños, como identificadores o nombres, se complica por la posibilidad de un **ataque de diccionario,** en particular si la clave de cifrado es pública. Por ejemplo, si se cifran los campos de fecha de nacimiento, un atacante intentando descifrar un determinado valor *e* puede intentar cifrar todos los posibles valores de fecha de nacimiento hasta que encuentre uno cuyo valor cifrado coincida con *e*. Incluso aunque la clave de cifrado no esté disponible públicamente, la información estadística sobre la distribución de los datos se puede usar para adivinar qué representa el valor cifrado en algunos casos, como la edad o el código postal. Por ejemplo, si la edad de 18 años es la edad más común en la base de datos, se podrá inferir que el valor cifrado de la edad que se produzca con más frecuencia representa 18.

Los ataques de diccionario se pueden combatir añadiendo bits extra aleatorios al final de cada uno de los valores antes de su cifrado, y eliminándolos al descifrar. Estos bits extra, llamados **vector de inicialización** en AES, o bits de *sal* en otros contextos, proporcionan un nivel de protección extra contra ataques de diccionario.

9.8.2. Soporte del cifrado en las bases de datos

Muchos sistemas de archivos y de bases de datos de hoy en día disponen del cifrado de los datos. Ese cifrado protege los datos de quien pueda tener acceso a ellos pero no sea capaz de tener acceso a la clave de descifrado. En el caso del cifrado del sistema de archivos, los datos que se cifran suelen ser archivos de gran tamaño y directorios que contienen información sobre los archivos.

En el contexto de las bases de datos, el cifrado se puede llevar a cabo en diferentes niveles. En el nivel inferior, se pueden cifrar los bloques de disco que contienen datos de la base de datos usando una clave disponible para el software del sistema de bases de datos. Cuando se recupera un bloque del disco, primero se descifra y luego se usa de la manera habitual. Este cifrado en el nivel de los bloques del disco protege contra los atacantes que puedan tener acceso al contenido del disco pero no dispongan de la clave de cifrado.

En el nivel siguiente, se pueden guardar cifrados ciertos atributos de una relación, o todos. En este caso, cada atributo de una relación podría tener una clave de cifrado diferente. Muchas bases de datos actuales permiten el cifrado en el nivel de atributos, así como en el nivel de una relación completa o de todas las relaciones de la base de datos. El cifrado de solo algunos atributos minimiza la sobrecarga del cifrado, permitiendo que las aplicaciones solo cifren los atributos con valores sensibles, como los números de las tarjetas de crédito. Sin embargo, cuando se cifran determinados atributos o relaciones, no se suele permitir que se cifren las claves primarias ni las claves externas, y no se permite el indizado de atributos cifrados. En el cifrado se necesita usar bits adicionales aleatorios para evitar un ataque de diccionario, como se ha descrito anteriormente.

Obviamente se necesita una clave de descifrado para acceder a los datos cifrados. Se puede usar una clave maestra única para todos los datos cifrados; con el cifrado de atributos, se pueden usar claves diferentes para atributos distintos. En este caso las distintas claves de cifrado se pueden guardar en un archivo o relación, normalmente llamado «wallet» (cartera), que a su vez se encuentra cifrado utilizando la clave maestra.

Cuando se realiza una conexión a la base de datos que necesita acceder a atributos cifrados, debe proporcionar la clave maestra; si no se proporciona la conexión no podrá acceder a los datos cifrados. La clave maestra se guarda en el programa de aplicación, normalmente en una computadora distinta, o memorizada en la base de datos de usuario, y se proporciona cuando este se conecta a la base de datos.

El cifrado en el nivel de la base de datos tiene la ventaja de requerir relativamente poca sobrecarga de tiempo y espacio, y no necesita la modificación de las aplicaciones. Por ejemplo, si se precisa proteger los datos de una base de datos en un portátil, se puede usar este tipo de cifrado. De forma similar, quien tenga acceso a las copias de seguridad de una base de datos no debería ser capaz de acceder a los datos de la copia de seguridad sin conocer la clave de descifrado.

Una alternativa a realizar el cifrado de la base de datos es realizarlo *antes* de que los datos se guarden en la base de datos. La aplicación realiza el cifrado de los datos antes de enviarlos a la base de datos y los descifra cuando los obtiene. Esta forma de cifrado requiere una modificación significativa de la aplicación, frente a la que realiza el sistema de base de datos.

9.8.3. Cifrado y autenticación

La autenticación basada en contraseñas la usan mucho los sistemas operativos y las bases de datos. Sin embargo, el uso de contraseñas tiene algunos inconvenientes, especialmente en las redes. Si un espía es capaz de «esnifar» los datos que se envían por la red, puede ser capaz de averiguar la contraseña cuando se envíe por la red. Una vez que el husmeador obtiene un usuario y una contraseña, se puede conectar a la base de datos fingiendo que es el usuario legítimo.

Un esquema más seguro es el sistema de **respuesta por desafío.** El sistema de bases de datos envía una cadena de desafío al usuario. El usuario cifra la cadena de desafío usando una contraseña secreta como clave de cifrado y devuelve el resultado. El sistema de bases de datos puede verificar la autenticidad del usuario descifrando la cadena con la misma contraseña secreta, y comparando el resultado con la cadena de desafío original. Este esquema garantiza que las contraseñas no viajan por la red.

Los sistemas de clave pública se pueden usar para el cifrado en los sistemas de respuesta por desafío. El sistema de bases de datos cifra la cadena de desafío usando la clave pública del usuario y se la envía al usuario. Éste descifra la cadena con su clave privada y devuelve el resultado al sistema de bases de datos. El sistema de bases de datos comprueba entonces la respuesta. Este esquema tiene la ventaja añadida de no almacenar la contraseña secreta en la base de datos, donde podrían verla los administradores del sistema.

Guardar la clave privada del usuario en una computadora, aunque sea una computadora personal, tiene el riesgo de que, si se pone en peligro la seguridad de la computadora, se puede revelar la clave al atacante, que podrá suplantar al usuario. Las **tarjetas inteligentes** proporcionan una solución a este problema. En las tarjetas inteligentes la clave se puede guardar en un chip incorporado; el sistema operativo de la tarjeta inteligente garantiza que no se pueda leer nunca la clave, pero permite que se envíen datos a la tarjeta para cifrarlos o descifrarlos usando la clave privada.[9]

9.8.3.1. Firma digital

Otra aplicación interesante del cifrado de clave pública está en la **firma digital** para comprobar la autenticidad de los datos; la firma digital desempeña el rol electrónico de las firmas físicas en los documentos. La clave privada se usa para «firmar»; es decir, cifrar los datos, y los datos firmados se pueden hacer públicos. Cualquiera puede comprobar la firma descifrando los datos con la clave pública, pero nadie puede haber generado los datos firmados sin tener la clave privada. Fíjese en la inversión de los roles de las claves pública y privada en este esquema. Por tanto, se pueden **autenticar** los datos; es decir, se puede comprobar que fueron creados realmente por la persona que afirma haberlos creado.

Además, las firmas digitales también sirven para garantizar el **no repudio.** Es decir, en el caso de que la persona que creó los datos afirmara posteriormente que no los creó, el equivalente electrónico de afirmar que no se ha firmado un cheque, se puede probar que esa persona tiene que haber creado los datos, a menos que su clave privada haya caído en manos de otros.

9.8.3.2. Certificados digitales

En general, la autenticación es un proceso de doble sentido en el que cada miembro del par de entidades que interactúa se autentica a sí mismo frente al otro. Esta autenticación por parejas es necesaria aunque un cliente entre en contacto con un sitio web, para

9 Las tarjetas inteligentes también proporcionan otra funcionalidad, como la capacidad de almacenar dinero digital y realizar pagos, lo que no es relevante en este contexto.

evitar que los sitios malévolos se hagan pasar por sitios web legales. Esa suplantación se puede llevar a cabo, por ejemplo, si se comprometen los encaminadores de la red y los datos se desvían al sitio malicioso.

Para que el usuario pueda estar seguro de que está interactuando con el sitio web auténtico debe tener la clave pública de ese sitio. Esto plantea el problema de hacer que el usuario consiga la clave pública, si se almacena en el sitio web, pudiendo el sitio malicioso proporcionar una clave diferente, con lo que el usuario no tendría manera de comprobar si la clave pública proporcionada es auténtica. La autenticación se puede conseguir mediante un sistema de **certificados digitales** en el que las claves públicas están firmadas por una agencia de certificación, cuya clave pública es bien conocida. Por ejemplo, las claves públicas de las autoridades de certificación raíz se almacenan en los navegadores web estándar. Los certificados emitidos por ellas se pueden comprobar mediante las claves públicas almacenadas.

Un sistema de dos niveles supondría una carga excesiva de creación de certificados para las autoridades de certificación raíz, por lo que se usa en su lugar un sistema de varios niveles, con una o más autoridades de certificación raíz y un árbol de autoridades de certificación por debajo de cada raíz. Cada autoridad, distinta de la autoridad raíz, tiene un certificado digital emitido por su autoridad padre.

El certificado digital emitido por la autoridad de certificación A consiste en una clave pública K_A y un texto cifrado E que se puede descifrar mediante la clave pública K_A. El texto cifrado contiene el nombre de la parte a la que se le ha emitido el certificado y su clave pública Kc. En el caso de que la autoridad de certificación A no sea autoridad de certificación raíz, el texto cifrado también contiene el certificado digital emitido a A por su autoridad de certificación padre; este certificado autentica la propia clave K_A (ese certificado puede, a su vez, contener un certificado de otra autoridad de certificación padre, y así sucesivamente).

Para comprobar un certificado, se descifra el texto cifrado E mediante la clave pública K_A para obtener el nombre de la parte, es decir, el nombre de la organización propietaria del sitio web; adicionalmente, si A no es autoridad raíz de quien se conozca la clave pública para el verificador, la clave pública K_A se comprueba de manera recursiva usando el certificado digital contenido en E; la recursividad termina cuando se llega a un certificado emitido por la autoridad raíz. La comprobación de los certificados establece la cadena mediante la cual se autentica cada sitio concreto y proporciona el nombre y la clave pública autenticada de ese sitio.

Los certificados digitales se usan mucho para autentificar los sitios web ante los usuarios, para evitar que sitios maliciosos suplanten a otros sitios web. En el protocolo HTTPS, la versión segura del protocolo HTTP, el sitio proporciona su certificado digital al navegador, que lo muestra entonces al usuario. Si el usuario acepta el certificado, el navegador usa la clave pública proporcionada para cifrar los datos. Los sitios maliciosos pueden tener acceso al certificado, pero no a la clave privada y, por tanto, no podrán descifrar los datos enviados por el navegador. Solo el sitio auténtico, que tiene la clave privada correspondiente, puede descifrar los datos enviados por el navegador. Hay que tener en cuenta que los costes del cifrado y descifrado con la clave pública y la clave privada son mucho más elevados que los de cifrado y descifrado mediante claves privadas simétricas. Para reducir los costes de cifrado, HTTPS crea realmente una clave simétrica de un solo uso tras la autenticación y la usa para cifrar los datos durante el resto de la sesión.

Los certificados digitales también se pueden usar para autenticar a los usuarios. El usuario debe remitir al sitio un certificado digital que contenga su clave pública; el sitio comprueba que el certificado haya sido firmado por una autoridad de confianza. Entonces se puede usar la clave pública del usuario en un sistema de respuesta por desafío para garantizar que ese usuario posee la clave privada correspondiente, lo que autentifica al usuario.

9.9. Resumen

- Los programas de aplicación usan las bases de datos como sistemas de soporte y su interacción con los usuarios ha existido desde los años sesenta. Las arquitecturas de aplicación han evolucionado en todo este tiempo. En la actualidad la mayoría de las aplicaciones usan navegadores web como *front-end*, y las bases de datos como *back-end*, con un servidor de aplicaciones entre medias.
- HTML tiene la capacidad de definir interfaces que combinan hipervínculos con formularios. Los navegadores web se comunican con los servidores web mediante el protocolo HTTP. Los servidores web pueden pasar solicitudes a los programas de aplicación y devolver el resultado al navegador.
- Los servidores web ejecutan programas de aplicación para implementar la funcionalidad deseada. Las servlets son un mecanismo muy usado para escribir programas de aplicación que se ejecutan como parte del proceso del servidor web, para reducir las sobrecargas. También hay muchos lenguajes de secuencias de comandos del lado del servidor que interpreta el servidor web y proporcionan la funcionalidad de los programas de aplicación como parte del servidor web.
- Hay varios lenguajes de secuencias de comandos del lado del cliente —JavaScript es el más usado— que proporcionan una mayor interacción con el usuario en el extremo del navegador.
- Las aplicaciones complejas suelen tener una arquitectura multicapa, incluyendo un modelo que implementa la lógica del negocio, un controlador y un mecanismo de vista para mostrar los resultados. También pueden incluir una capa de acceso a los datos que implementa la correspondencia objeto-relación. Muchas aplicaciones implementan y usan servicios web, que permiten invocar las funciones usando HTTP.
- Se han desarrollado herramientas para el desarrollo rápido de aplicaciones, y en particular para reducir el esfuerzo que requiere construir las interfaces gráficas de usuario.
- Para mejorar el rendimiento de las aplicaciones se utilizan técnicas de caché en varias formas, incluyendo el caché de resultados, los bancos de conexiones y el procesamiento paralelo.
- Los desarrolladores de aplicaciones deben prestar mucha atención a la seguridad, para evitar ataques como la inyección de SQL y otros ataques de secuencias de comandos de sitios cruzados.
- Los mecanismos de autorización de SQL son de grano grueso y de un valor limitado para las aplicaciones que tratan con un gran número de usuarios. En la actualidad, los programas de aplicaciones implementan la autorización de grano fino, en el nivel de tupla, tratando con un gran número de usuarios, completamente fuera del sistema de bases de datos. Se han desarrollado extensiones para proporcionar nivel de acceso al nivel de tupla y para tratar con un gran número de aplicaciones, pero aún no están estandarizados.

- La protección de la privacidad de los datos es una importante tarea de las aplicaciones de bases de datos. Muchos países tienen establecidos requisitos legales sobre la protección de ciertos tipos de datos, como la información de tarjetas de crédito o de datos médicos.
- El cifrado desempeña un papel fundamental en la protección de la información y en la autenticación de los usuarios y de los sitios web. El cifrado de clave simétrica y el cifrado de clave pública son técnicas de cifrado contrastadas y muy extendidas. El cifrado de ciertos datos sensibles almacenados en las bases de datos es un requisito legal en muchos países.
- El cifrado también desempeña un papel fundamental en la autenticación de usuarios a las aplicaciones, de sitios web a los usuarios y para la firma digital.

Términos de repaso

- Programas de aplicación.
- Interfaces web con bases de datos.
- Hiperenlaces.
- Localizador uniforme de recursos (URL).
- Formularios.
- Protocolo de transferencia de hipertexto (HTTP).
- Interfaz de pasarela común (CGI).
- Protocolos sin conexión.
- Cookie.
- Sesión.
- Servlets y sesiones de Servlet.
- Secuencia de comandos del lado del servidor.
- JSP.
- PHP.
- ASP.NET.
- Secuencia de comandos del lado del cliente.
- JavaScript.
- Modelo de objetos de documentos (DOM).
- Applets.
- Arquitectura de aplicación.
- Capa de presentación.
- Arquitectura modelo-vista-controlador (MVC).
- Capa de la lógica de negocio.
- Capa de acceso a los datos.
- Correspondencia objeto-relación.
- Hibernate.
- Lenguaje de marcado de hipertexto (HTML).
- Servicios web.
- Servicios RESTful.
- Desarrollo rápido de aplicaciones.
- Framework de aplicaciones web.
- Generadores de informes.
- Banco de conexiones.
- Caché de resultados de consultas.
- Seguridad de las aplicaciones.
- Inyección SQL.
- Secuencias de comandos de sitios cruzados (XSS).
- Falsificación de petición en sitios cruzados (XSRF).
- Autenticación.
- Autenticación en dos pasos.
- Ataque de hombre en el medio.
- Autenticación central.
- Registro de firma única.
- OpenID.
- Base de datos privada virtual (VPD).
- Trazas de auditoría.
- Cifrado.
- Cifrado de clave simétrica.
- Cifrado de clave pública.
- Ataque de diccionario.
- Desafío-respuesta.
- Firma digital.
- Certificado digital.

Ejercicios prácticos

9.1. ¿Cuál es la razón principal de que los servlets den mejor rendimiento que los programas que usan la interfaz de pasarela común (common gateway interface, CGI), pese a que los programas de Java suelen ejecutarse más lentamente que los programas de C o de C++?

9.2. Indique algunas de las ventajas y de los inconvenientes de los protocolos sin conexión frente a los protocolos que mantienen las conexiones.

9.3. Considere una aplicación web, escrita sin mucho cuidado, para un sitio de compra online que almacena el precio de cada artículo en una variable oculta de un formulario en la página web que envía al cliente; cuando el cliente envía el formulario, se usa la información de la variable oculta del formulario para calcular la factura del cliente. ¿Qué agujero tiene este esquema? Existió un ejemplo real en el que el agujero de seguridad fue explotado por algunos usuarios de sitio de compra online antes de que el problema se detectase y reparase.

9.4. Considere otra aplicación web escrita de manera poco cuidadosa, que usa un servlet que comprueba si existe una sesión activa, pero no comprueba si el usuario está autorizado para acceder a la página, sino que solo depende de que el enlace se muestra exclusivamente a usuarios autorizados. ¿Cuál es el riesgo de este esquema? (Existió un ejemplo real en el que los que deseaban la admisión en una universidad podían, tras registrarse en el sitio web, explotar este agujero de seguridad para ver información que no estaban autorizados a ver; el acceso no autorizado se detectó y a aquellos que accedieron a la información se les penalizó no admitiéndoles en la universidad.)

9.5. Indíquense tres maneras en que se puede usar el almacenamiento en caché para acelerar el rendimiento de los servidores web.

9.6. El comando `netstat` (disponible tanto en Linux como en Windows) muestra las conexiones de red activas en una computadora. Explique cómo se puede usar este comando para

descubrir si una página web en particular no está cerrando las conexiones que abre, o si se usa un banco de conexiones y no devuelve las conexiones al banco. Debería tener en cuenta que con el banco de conexiones las conexiones no se cierran inmediatamente.

9.7. Comprobando una vulnerabilidad de inyección SQL:

a. Sugiera una forma de comprobar en una aplicación si es vulnerable a ataques de inyección SQL en la entrada de texto.

b. ¿Puede producirse una inyección SQL con otras formas de entrada? Si es así, cómo se podría comprobar esta vulnerabilidad?

9.8. Suponga una relación de una base de datos en la que los valores de ciertos atributos se han cifrado por seguridad. ¿Por qué los sistemas de bases de datos no permiten el indizado sobre atributos cifrados? Use su respuesta a la pregunta anterior para explicar por qué los sistemas de bases de datos no permiten el cifrado de atributos de clave primaria.

9.9. En el Ejercicio 9.8 se trata el problema del cifrado de ciertos atributos. Sin embargo, algunos sistemas de bases de datos permiten el cifrado de toda la base de datos. Explique por qué se evitan los problemas planteados en el Ejercicio 9.8 cuando se cifra toda la base de datos.

9.10. Suponga que alguien suplanta a una empresa y obtiene un certificado de una autoridad emisora de certificados. ¿Cuál es el efecto sobre las cosas (como los pedidos de compras o los programas) certificadas por la empresa suplantada y sobre las cosas certificadas por otras empresas?

9.11. Quizás los elementos de datos más importantes de cualquier sistema de bases de datos sean las contraseñas que controlan el acceso a la base de datos. Proponga un esquema para guardar de manera segura las contraseñas. Es necesario asegurar que el esquema permite al sistema comprobar las contraseñas proporcionadas por los usuarios que intenten iniciar una sesión en el mismo.

Ejercicios

9.12. Escriba un servlet y el código HTML asociado para la siguiente aplicación, muy sencilla: se permite que los usuarios remitan un formulario que contiene un valor, por ejemplo, *n*, y deben obtener una respuesta que contenga *n* símbolos "*".

9.13. Escriba un servlet y el código HTML asociado para la siguiente aplicación sencilla: se permite que los usuarios remitan un formulario que contiene un número, por ejemplo *n*, y deben obtener una respuesta que indique el número de veces que se ha remitido anteriormente el valor *n*. El número de veces que se ha remitido anteriormente el valor se debe almacenar en una base de datos.

9.14. Escriba un servlet que autentique a los usuarios (de acuerdo con los nombres de usuario y las contraseñas almacenadas en una relación de una base de datos) y defina una variable de sesión denominada *idusuario* tras la autenticación.

9.15. ¿Qué son los ataques de inyección de SQL? Explique cómo funcionan y las precauciones que se deben adoptar para evitarlos.

9.16. Escriba un pseudocódigo para gestionar un banco de conexiones. El pseudocódigo debe incluir una función que cree el banco (proporcionando una cadena de caracteres de conexión a la base de datos, el nombre de usuario de la base de datos y la contraseña como parámetros), una función que solicite una conexión al banco, una función que libere una conexión al banco y una función que cierre el banco de conexiones.

9.17. Explique los términos CRUD y REST.

9.18. Muchos sitios web actuales proporcionan ricas interfaces de usuario usando Ajax. Indique dos características que pueden revelar si un sitio usa Ajax, sin tener que ver el código fuente. Usando las características anteriores, descubra tres sitios web que usan Ajax; puede ver el código HTML de la página para comprobar si el sitio usa realmente Ajax.

9.19. Sobre los ataques XSS:

a. ¿Qué es un ataque XSS?

b. ¿Cómo se puede usar el campo «referer» para detectar algunos ataques XSS?

9.20. ¿Qué es la autenticación multipaso? ¿Cómo ayuda a protegerse contra el robo de contraseñas?

9.21. Considere la característica de **base de datos privada virtual** (**VPD**) de Oracle que se describe en la Sección 9.7.5 y una aplicación basada en nuestro esquema de universidad.

a. ¿Qué predicado (usando una subconsulta) se debería generar para permitir que los miembros de una facultad vean solo las tuplas de *matricula* que corresponden a secciones de asignaturas que ellos han enseñado?

b. Escriba una consulta de SQL tal que la consulta con el predicado añadido genere un resultado que es un subconjunto de la consulta resultado original sin añadir el predicado.

c. Escriba una consulta de SQL tal que la consulta con el predicado añadido genere un resultado que contiene una tupla que no está en el resultado original de la consulta sin añadir el predicado.

9.22. Indique dos ventajas de cifrar los datos guardados en una base de datos.

9.23. Suponga que se desea crear una traza de auditoría de las modificaciones de la relación *matricula*.

a. Defina los disparadores para crear una traza de auditoría y registrar la información en una relación denominada, por ejemplo, *traza_matricula*. La información registrada debe incluir el identificador de usuario (suponga que la función *id_usuario*() proporciona esa información) y una marca de tiempo, además de los valores antiguos y nuevos. También hay que proporcionar el esquema de la relación *traza_matricula*.

b. ¿Puede la implementación anterior garantizar que las actualizaciones realizadas por administradores maliciosos de la base de datos (o por alguien que consiga la contraseña del administrador) se hallen en la traza de auditoría? Explique su respuesta.

9.24. Los hackers pueden lograr hacer creer al usuario que su sitio web es realmente un sitio web en que el usuario confía (como el de un banco o el de una tarjeta de crédito). Esto puede lograrse mediante mensajes engañosos de correo electrónico o, incluso, mediante la reconfiguración de la infraestructura de la red y el redireccionamiento del tráfico de red destinado a, por ejemplo, mibanco.com, hacia el sitio de los hackers. Si se introduce el nombre de usuario y la contraseña en el sitio de los hackers, ese sitio puede registrarlo y usarlo posteriormente para entrar en la cuenta en el sitio verdadero. Cuando se usa una URL como https://mibanco.com, se usa el protocolo HTTPS para evitar esos ataques. Explique la manera en que el protocolo puede usar los certificados digitales para comprobar la autenticidad del sitio.

9.25. Explique qué es el sistema de autenticación de respuesta por desafío. ¿Por qué es más seguro que los sistemas tradicionales basados en las contraseñas?

Sugerencias de proyectos

Cada uno de los proyectos que aparecen a continuación es un proyecto de grandes dimensiones. La dificultad de cada proyecto puede ajustarse fácilmente añadiendo o eliminando características.

Proyecto 9.1 Elija su sitio web interactivo favorito, como Bebo, Blogger, Facebook, Flickr, Last.FM, Twitter, Wikipedia; son solo algunos ejemplos, hay muchos más. La mayoría de estos sitios gestionan una gran cantidad de datos y usan bases de datos para almacenarlos y procesarlos. Implemente un subconjunto de la funcionalidad del sitio web que haya elegido. Claramente, implementar solamente un subconjunto significativo de las características de un sitio queda más allá del proyecto de un curso, pero es posible definir un conjunto de características que resulte interesante implementar, pero suficientemente pequeño para un proyecto de curso.

La mayoría de los sitios web más populares usan extensamente JavaScript para crear interfaces. No sea muy ambicioso inicialmente en su proyecto, al menos inicialmente, ya que requiere mucho tiempo construir esas interfaces y después añadir las característica a las mismas. Utilice un framework de desarrollo de aplicaciones web, o bibliotecas de JavaScript disponibles en la web, como la biblioteca Yahoo User Interface, para acelerar el desarrollo.

Proyecto 9.2 Cree un «mashup» que use servicios como las API de mapas de Google o de Yahoo para crear un sitio web interactivo. Por ejemplo, la API de mapas proporciona una forma de mostrar un mapa en una página web, en la que se puede superponer otra información. Podría implementar un sistema de recomendación de restaurantes en el que los usuarios contribuyen con información sobre los restaurantes, como su ubicación, tipo de cocina, intervalo de precios y valoración. Los resultados de búsquedas de los usuarios se podrían mostrar en el mapa. Podría añadir características del tipo de Wikipedia, como que los usuarios puedan añadir información y editar la que han añadido otros usuarios, junto con moderadores que puedan realizar revisiones frente a actualizaciones malintencionadas. También podría implementar características sociales, como dar más importancia a las valoraciones que hacen los amigos.

Proyecto 9.3 Su universidad probablemente use un sistema de gestión de cursos como Moodle, Blackboard o WebCT. Implemente un subconjunto de la funcionalidad de un sistema de gestión de cursos. Por ejemplo, podría proporcionar el envío de tareas y su corrección, incluyendo mecanismos para que los estudiantes y los profesores hablen sobre las notas de una determinada tarea. También podría implementar un mecanismo para realizar encuestas para obtener realimentación.

Proyecto 9.4 Considere el esquema E-R del Ejercicio práctico 7.3 (Capítulo 7), que representa la información de los equipos de una liga. Diseñe e implemente un sistema basado en web para introducir, actualizar y examinar los datos.

Proyecto 9.5 Diseñe e implemente un sistema de carro de la compra que permita a los compradores reunir artículos en un carro (decida la información que se proporciona de cada artículo) y comprarlos todos juntos. Puede extender y usar el esquema E-R del Ejercicio 7.20 del Capítulo 7. Conviene comprobar la disponibilidad del artículo y tratar los artículos no disponibles del modo que se considere adecuado.

Proyecto 9.6 Diseñe e implemente un sistema basado en web para registrar la información sobre las matrículas y las notas de los estudiantes para los cursos de una universidad.

Proyecto 9.7 Diseñe e implemente un sistema que permita el registro de la información de rendimiento de las asignaturas; concretamente, las notas otorgadas a cada estudiante en cada trabajo o examen de cada asignatura, y el cálculo de una suma (ponderada) de las notas para obtener las notas totales de la asignatura. El número de trabajos o exámenes no debe estar predefinido; es decir, se pueden añadir más trabajos o exámenes en cualquier momento. El sistema también debe soportar la clasificación, permitiendo que se especifiquen separadores para varias notas.

Puede que también se desee integrarlo en el sistema de matrícula de los estudiantes del Proyecto 9.6 (que quizás implemente otro equipo de proyecto).

Proyecto 9.8 Diseñe e implemente un sistema basado en web para la reserva de aulas de una universidad. Se debe soportar la reserva periódica (días y horas fijas cada semana para un semestre completo). También debe soportar la cancelación de aulas concretas de una reserva periódica.

Puede que también se desee integrarlo con el sistema de matrícula de estudiantes del Proyecto 9.6 (que quizás implemente otro equipo de proyecto) de modo que las aulas puedan reservarse para asignaturas y las cancelaciones de una clase o las reservas de clases adicionales puedan anotarse en una sola interfaz, se reflejen en la reserva de aulas y se comuniquen a los estudiantes por correo electrónico.

Proyecto 9.9 Diseñe e implemente un sistema para gestionar exámenes de tipo test (con varias respuestas posibles) en línea. Se debe soportar el aporte distribuido de preguntas (por los profesores ayudantes, por ejemplo), la edición de las preguntas por quien esté a cargo de la asignatura y la creación de exámenes a partir del conjunto de preguntas disponible. También se deben poder suministrar los exámenes en línea, bien a una hora fija para todos los estudiantes o a cualquier hora pero con un límite de tiempo desde el comienzo hasta el final (se pueden soportar las dos opciones o solo una de ellas) y dar a los estudiantes información sobre su puntuación al final del tiempo concedido.

Proyecto 9.10 Diseñe e implemente un sistema para gestionar el servicio de correo electrónico de los clientes. El correo entrante va a un buzón común. Hay una serie de agentes de atención al cliente que responden el correo electrónico. Si el mensaje es parte de una serie de respuestas (que se siguen mediante el campo *responder a* del correo electrónico) es preferible que responda el mensaje el mismo agente que lo haya respondido anteriormente. El sistema debe realizar un seguimiento de todo el correo entrante y de las respuestas, de modo que los agentes puedan ver el historial de preguntas de cada cliente antes de responder a los mensajes.

Proyecto 9.11 Diseñe e implemente un mercado electrónico sencillo en el que los artículos puedan clasificarse para su compra o venta en varias categorías (que deben formar una jerarquía). Puede que también se deseen soportar los servicios de alerta, en los que los usuarios pueden registrar su interés por los artículos de una categoría concreta, quizás con otras restricciones añadidas, sin anunciar su interés de manera pública; después, cuando alguno de esos artículos se ofrece a la venta, se les notifica.

Proyecto 9.12 Diseñe e implemente un sistema de grupos de noticias basado en web. Los usuarios deben poder suscribirse a los grupos de noticias y leer los artículos de esos grupos. El sistema hace un seguimiento de los artículos que el usuario ha leído, de modo que no vuelvan a mostrarse. También hay que permitir la búsqueda de artículos antiguos. Es posible que también se desee ofrecer un servicio de valoración de los artículos, de modo que los artículos con puntuación elevada se destaquen, lo que permite al lector ocupado saltarse los artículos con baja puntuación.

Proyecto 9.13 Diseñe e implemente un sistema basado en web para gestionar una clasificación deportiva. Se registra mucha gente y se les da alguna clasificación inicial (quizá basada en resultados históricos). Cualquier usuario puede retar a cualquier otro y las clasificaciones se ajustan según los resultados. Un sistema sencillo para ajustar las clasificaciones se limita a pasar al ganador por delante del perdedor en la clasificación, en caso de que el ganador estuviera por detrás. Se puede intentar inventar sistemas de ajustes de la clasificación más complicados.

Proyecto 9.14 Diseñe e implemente un servicio de listados de publicaciones. El servicio debe permitir la introducción de información sobre las publicaciones, como pueden ser título, autores, año en que apareció la publicación, páginas, etc. Los autores deben ser una entidad separada con atributos como nombre, institución, departamento, correo electrónico, dirección y página web.

La aplicación debe soportar varias vistas de los mismos datos. Por ejemplo, se deben facilitar todas las publicaciones de un autor dado (ordenadas por año, por ejemplo), o todas las publicaciones de los autores de una institución o de un departamento dado. También se debe soportar la búsqueda por palabras clave, tanto en la base de datos global como en cada una de las vistas.

Proyecto 9.15 Una tarea frecuente en cualquier organización es la recogida de información estructurada de un grupo de personas. Por ejemplo, puede que un director necesite pedir a los empleados que introduzcan sus planes de vacaciones, un profesor puede que desee obtener la respuesta de los estudiantes a un tema concreto, el estudiante que organiza un evento puede que desee permitir que otros estudiantes se inscriban en él o alguien puede desear organizar una votación en línea sobre un tema concreto.

Cree un sistema que permita a los usuarios crear con facilidad eventos de reunión de información. Al crear un evento, su creador debe definir quién tiene derecho a participar; para ello, el sistema debe conservar la información de los usuarios y permitir predicados que definan subconjuntos de usuarios. El creador del evento debe poder especificar conjuntos de datos (con tipos, valores predeterminados y controles de validación) que tendrán que proporcionar los usuarios. El evento debe tener una fecha límite asociada y la posibilidad de enviar recordatorios a los usuarios que todavía no hayan remitido su información. Se puede dar al creador del evento la opción de hacer que se cumpla la fecha límite de manera automática con base en una fecha u hora específicas, o que decida iniciar una sesión y declarar que se ha superado la fecha límite. Se deben generar estadísticas sobre los datos enviados, para ello se permitirá al creador del evento que cree resúmenes sencillos de la información aportada. El creador del evento puede elegir hacer públicos parte de los resúmenes, visibles a todos los usuarios, de manera continua (por ejemplo, el número de personas que ha respondido) o una vez superada la fecha límite (por ejemplo, la puntuación promedio obtenida).

Proyecto 9.16 Cree una biblioteca de funciones para simplificar la creación de interfaces web. Hay que implementar, como mínimo, las siguientes funciones: una función que muestre el conjunto de resultados JDBC (en formato tabular), funciones que creen diferentes tipos de datos de texto y numéricos (con criterios de validación como el tipo de entrada y un intervalo opcional, aplicado en el cliente mediante el código Javascript correspondiente), funciones que introduzcan los valores de la fecha y de la hora (con valores predeterminados) y funciones que creen elementos de menú de acuerdo con un conjunto de resultados. Para obtener más nota se debe permitir al usuario configurar parámetros de estilo, como los colores y las fuentes, y proporcionar soporte a la paginación en las tablas (se pueden usar parámetros ocultos para los formularios para especificar la página que se debe mostrar). Cree una aplicación de bases de datos de ejemplo para ilustrar el uso de estas funciones.

Proyecto 9.17 Diseñe e implemente un sistema de agenda multiusuario basado en web. El sistema debe realizar un seguimiento de las citas de cada persona, con eventos de ocurrencia múltiple, como reuniones semanales, eventos compartidos (en los que las actualizaciones realizadas por el creador del evento se reflejan en las agendas de todos los que comparten ese evento). Proporciónese la notificación de los eventos por correo electrónico. Para obtener más nota se debe implementar un servicio web que pueda usar un programa de recordatorios que se ejecute en la máquina cliente.

Herramientas

El desarrollo de aplicaciones web necesita distintas herramientas de software como servidores de aplicaciones, compiladores y editores de lenguajes de programación como Java o C#, y otras herramientas opcionales como los servidores web. Existen distintos entornos de desarrollo integrado que proporcionan soporte para el desarrollo de aplicaciones web. Los dos IDE de código abierto más populares son Eclipse, desarrollado por IBM, y Netbeans, desarrollado por Sun Microsystems. Visual Studio de Microsoft es el IDE más usado en mundo Windows.

Apache Tomcat (jakarta.apache.org), Glassfish (glassfish.dev.java.net), JBoss (jboss.org) y Caucho's Resin (www.caucho.com), son servidores de aplicaciones que soportan servlets y JSP. El servidor web Apache (apache.org) es el servidor web más usado en la actualidad. IIS (Internet Information Services) de Microsoft es un servidor web y de aplicaciones ampliamente usado en plataformas de Microsoft Windows que soporta ASP.NET (msdn.microsoft.com/asp.net/) de Microsoft.

El software WebSphere (www.software.ibm.com) de IBM proporciona un conjunto de herramientas para el desarrollo y despliegue de aplicaciones web, incluyendo un servidor de aplicaciones, un IDE, integración de aplicaciones middleware, software de gestión de procesos de negocio y herramientas de administración de sistemas.

Algunas de las herramientas anteriores son de código abierto que se pueden usar de forma gratuita, algunas son gratuitas para uso no comercial o uso personal, mientras que otras son de pago. Véanse los sitios web respectivos para más información.

La biblioteca de JavaScript Yahoo! User Interface (YUI) (developer.yahoo.com/yui) se usa extensamente para la creación de programas en JavaScript que se ejecutan en distintos navegadores.

Notas bibliográficas

La información sobre los servlets, incluidos los tutoriales, las especificaciones de las normas y el software están disponibles en java.sun.com/products/servlet. La información sobre JSP está disponible en java.sun.com/products/jsp. La información sobre las bibliotecas de etiquetas de JSP también se puede encontrar en esa URL. La información sobre la estructura .NET y sobre el desarrollo de aplicaciones web mediante ASP.NET se puede hallar en msdn.microsoft.com.

Atreya et ál. [2002] proporcionan un tratamiento del nivel de los libros de texto de las firmas digitales, incluidos los certificados digitales X.509 y de la infraestructura de clave pública.

Parte 3
Almacenamiento de datos y consultas

Aunque los sistemas de bases de datos proporcionan una visión de alto nivel de los datos, en último término es necesario guardarlos como bits en uno o varios dispositivos de almacenamiento. Una amplia mayoría de sistemas de bases de datos actuales almacenan los datos en discos magnéticos (y de forma creciente en almacenamiento flash) y los extraen a la memoria principal para su procesamiento, o los copian como archivos en cintas u otros dispositivos de copia de seguridad. Las características físicas de los dispositivos de almacenamiento desempeñan un papel importante en el modo en que se almacenan los datos, en especial porque el acceso a un fragmento aleatorio de los datos en el disco resulta mucho más lento que el acceso a la memoria: los accesos al disco invierten decenas de milisegundos, mientras que el acceso a la memoria invierte una décima de microsegundo.

En el Capítulo 10 se comienza con una introducción a los medios físicos de almacenamiento, incluidos los mecanismos para minimizar las posibilidades de pérdida de datos debidas a fallos de los dispositivos. A continuación se describe el modo en que se asignan los registros a los archivos que, a su vez, se asignan a bits del disco.

Muchas consultas solo hacen referencia a una pequeña parte de los registros de un archivo. Los índices son estructuras que ayudan a localizar rápidamente los registros deseados de una relación, sin tener que examinar todos sus registros. El índice de este libro de texto es un ejemplo de ello aunque está pensado para su empleo por personas, a diferencia de los índices de las bases de datos. En el Capítulo 12 se describen varios tipos de índices utilizados en los sistemas de bases de datos.

Las consultas de los usuarios tienen que ejecutarse sobre el contenido de la base de datos, que reside en los dispositivos de almacenamiento. Suele ser conveniente dividir las consultas en operaciones más pequeñas, que se correspondan aproximadamente con las operaciones del álgebra relacional. En el Capítulo 12 se describe el modo en que se procesan las consultas, presenta los algoritmos para la implementación de las operaciones individuales y esboza el modo en que las operaciones se ejecutan en sincronía para procesar una consulta.

Existen muchas maneras alternativas de procesar cada consulta con costes muy distintos. La optimización de consultas hace referencia al proceso de hallar el método de menor coste para evaluar una consulta dada. En el Capítulo 13 se describe este proceso de optimización de consultas.

Almacenamiento y estructura de archivos

10

En los capítulos anteriores se han estudiado los modelos de bases de datos de alto nivel. Por ejemplo, en el nivel *conceptual* o *lógico* se ha presentado una base de datos del modelo relacional como un conjunto de tablas. En realidad, el modelo lógico de las bases de datos es el mejor nivel para que se centren los *usuarios*. Esto se debe a que el objetivo de los sistemas de bases de datos es simplificar y facilitar el acceso a los datos; no se debe agobiar innecesariamente a quienes utilizan el sistema con los detalles físicos de su implementación.

En este capítulo, no obstante, así como en los Capítulos 11, 12 y 13, se analizan niveles inferiores y se describen diferentes métodos de implementación de los modelos de datos y de los lenguajes presentados en capítulos anteriores. Se comienza con las características de los medios de almacenamiento subyacentes, como sistemas de disco y de cinta. Más adelante se definen varias estructuras de datos que permiten un acceso rápido a los datos. Se consideran varias arquitecturas alternativas, idóneas para diferentes tipos de acceso a los datos. La elección final de la estructura de datos hay que hacerla en función del uso que se espera dar al sistema y de las características de cada máquina concreta.

10.1. Visión general de los medios físicos de almacenamiento

La mayoría de los sistemas informáticos presentan varios tipos de almacenamientos de datos. Estos medios se clasifican según la velocidad con la que se puede tener acceso a los datos, su coste de adquisición por unidad de datos y su fiabilidad. Entre los medios disponibles habitualmente figuran:

- **Caché.** La caché es el mecanismo de almacenamiento más rápido y costoso. La memoria caché es pequeña; su uso lo gestiona el hardware del sistema informático. No debemos preocuparnos de la gestión del almacenamiento en caché en el sistema de base de datos. Sin embargo, hay que indicar que los implementadores de la base de datos prestan atención a los efectos de la caché cuando diseñan las estructuras de datos y los algoritmos de procesamiento de consultas.
- **Memoria principal.** El medio de almacenamiento utilizado para los datos con los que se opera es la memoria principal. Las instrucciones máquina operan en la memoria principal. Aunque la memoria principal puede contener varios gigabytes de datos en una computadora personal o incluso cientos de gigabytes en grandes sistemas servidor, suele ser demasiado pequeña (o demasiado cara) para guardar toda la base de datos. El contenido de la memoria principal suele perderse en caso de fallo del suministro eléctrico o de caída del sistema.
- **Memoria flash.** La memoria flash se diferencia de la memoria principal en que los datos no se pierden en caso de que se produzca un corte de la alimentación. Existen dos tipos de memorias flash, llamadas NAND y NOR. De ellas, las memorias flash NAND tienen mucha mayor capacidad de almacenamiento para un coste dado y se suelen usar para el almacenamiento de datos en dispositivos como cámaras, reproductores de música y teléfonos móviles, y cada vez más en computadoras portátiles. Las memorias flash tienen menor coste por byte que la memoria principal, además de no ser volátiles, es decir, mantienen los datos guardados aunque se pierda la alimentación.

  La memoria flash también se usa extensamente para guardar datos en las «llaves USB», que se conectan en un puerto *bus* serie universal (USB: *universal serial bus*) de la computadora. Estas llaves USB se han hecho muy populares para llevar información de un equipo a otro (los disquetes jugaban el mismo papel antes, pero su limitada capacidad ha hecho que se consideren obsoletos en la actualidad).

  La memoria flash también va aumentando su uso sustituyendo a los discos magnéticos para guardar una moderada cantidad de datos. Estos discos se llaman *unidades de estado sólido*. En 2009, un disco de estado sólido de 64 GB costaba menos de 200 € y su capacidad es de hasta 160 GB. Las memorias flash se usan cada vez más en sistemas servidor para mejorar el rendimiento realizando caché de los datos más utilizados, ya que proporcionan un acceso más rápido que el disco, con mayor capacidad de almacenamiento que la memoria principal (para un coste dado).
- **Almacenamiento en discos magnéticos.** El principal medio de almacenamiento persistente de los datos es el disco magnético. Generalmente se guarda en ellos toda la base de datos. Para acceder a los datos es necesario trasladarlos desde el disco a la memoria principal. Después de realizar la operación deseada se deben escribir en el disco los datos que se hayan modificado.

  En 2009, el tamaño de los discos magnéticos variaba entre unos 80 gigabytes y 1,5 terabytes. Un disco de 1 terabyte tenía un coste de unos 100 €. La capacidad de los discos ha ido creciendo a un ritmo cercano al cincuenta por ciento anual, y cada año se pueden esperar discos de mucha mayor capacidad. El almacenamiento en disco resiste los fallos del suministro eléctrico y las caídas del sistema. Los propios dispositivos de almacenamiento en disco pueden fallar a veces y, en consecuencia, destruir los datos, pero estos fallos se producen con mucha menos frecuencia que las caídas del sistema.
- **Almacenamiento óptico.** La forma más popular de almacenamiento óptico es el *disco compacto* (CD: *compact disk*), que puede almacenar alrededor de 700 megabytes de datos y tiene un tiempo de reproducción de 80 minutos, y el *disco de vídeo digital* (DVD: *digital video disk*), que puede almacenar 4,7 u 8,5

gigabytes de datos en cada cara del disco (o hasta 17 gigabytes en un disco de doble cara). También se utiliza el término **disco digital versátil** en lugar de **disco de vídeo digital,** ya que los DVD pueden almacenar cualquier tipo de dato digital, no solo datos de vídeo. Los datos se almacenan ópticamente en el disco y se leen mediante un láser. Un formato de mayor capacidad denominado *Blu-ray DVD* puede almacenar 27 gigabytes por cara o 54 gigabytes en un disco de doble cara.

No se puede escribir en los discos ópticos empleados como discos compactos de solo lectura (CD-ROM) o como discos de vídeo digital de solo lectura (DVD-ROM), pero se suministran con datos pregrabados. Existen también versiones «para una sola grabación» de los discos compactos (denominados CD-R) y de los discos de vídeo digital (DVD-R y DVD+R) en los que solo se puede escribir una vez; estos discos también se denominan **de escritura única y lectura múltiple** (WORM: *write-once, read-many*). Existen también versiones «para escribir varias veces» de los discos compactos (CD-RW) y de los discos de vídeo digital (DVD-RW, DVD+RW y DVD-RAM), en los que se puede escribir varias veces.

Los **cambiadores automáticos (jukebox)** de discos ópticos contienen varias unidades y numerosos discos que pueden cargarse de manera automática en las diferentes unidades (mediante un brazo robotizado) a petición del usuario.

- **Almacenamiento en cinta.** El almacenamiento en cinta se utiliza principalmente para copias de seguridad y datos archivados. Aunque la cinta magnética es más barata que los discos, el acceso a los datos resulta mucho más lento, ya que hay que acceder secuencialmente desde el comienzo de la cinta. Por este motivo, se dice que el almacenamiento en cinta es de **acceso secuencial,** mientras que el almacenamiento en disco es de **acceso directo,** ya que se pueden leer datos de cualquier parte del disco.

  Las cintas poseen una capacidad elevada (actualmente se dispone de cintas de 40 a 300 gigabytes) y pueden retirarse de la unidad de lectura, por lo que resultan idóneas para el almacenamiento de archivos con coste reducido. Los cambiadores automáticos de cinta (jukeboxes) se utilizan para guardar conjuntos de datos excepcionalmente grandes, como los datos obtenidos mediante satélite, que pueden ocupar centenares de terabytes (10^{12} bytes), o incluso petabytes (10^{15} bytes) en algunos casos.

Los diferentes medios de almacenamiento se pueden organizar de forma jerárquica (Figura 10.1) de acuerdo con su velocidad y su coste. Los niveles superiores resultan caros, pero son rápidos. A medida que se desciende por la jerarquía disminuye el coste por bit, pero aumenta el tiempo de acceso. Este compromiso es razonable; si un sistema de almacenamiento dado fuera a la vez más rápido y menos costoso que otro (en igualdad del resto de condiciones) no habría ninguna razón para utilizar la memoria más lenta y más cara. De hecho, muchos dispositivos de almacenamiento primitivos, como la cinta de papel y las memorias de núcleos de ferrita, se hallan relegados a los museos ahora que la cinta magnética y la memoria de semiconductores son más rápidas y baratas. Las propias cintas magnéticas se utilizaban para guardar los datos activos cuando los discos resultaban costosos y tenían una capacidad de almacenamiento reducida. Hoy en día casi todos los datos activos se almacenan en disco, excepto en los casos excepcionales en que se guardan en cambiadores automáticos de cinta u ópticos.

Los medios de almacenamiento más rápidos (por ejemplo, la caché y la memoria principal) se denominan **almacenamiento primario.** Los medios del siguiente nivel de la jerarquía (por ejemplo, los discos magnéticos) se conocen como **almacenamiento secundario** o **almacenamiento en conexión.** Los medios del nivel inferior de la jerarquía —por ejemplo, la cinta magnética y los cambiadores automáticos de discos ópticos— se denominan **almacenamiento terciario** o **almacenamiento sin conexión.**

Además de la velocidad y del coste de los diferentes sistemas de almacenamiento hay que tener en cuenta la volatilidad del almacenamiento. El **almacenamiento volátil** pierde su contenido cuando se suprime el suministro eléctrico del dispositivo. En la jerarquía mostrada en la Figura 10.1 los sistemas de almacenamiento desde la memoria principal hacia arriba son volátiles, mientras que los sistemas de almacenamiento por debajo de la memoria principal son no volátiles. Los datos se deben escribir en **almacenamiento no volátil** por motivos de seguridad. Este asunto se volverá a tratar en el Capítulo 16.

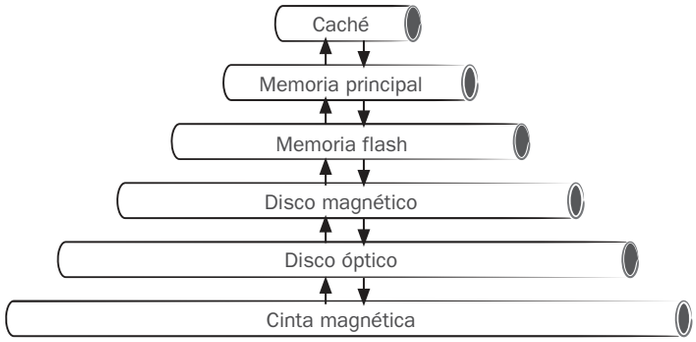


Figura 10.1. Jerarquía de los dispositivos de almacenamiento.

10.2. Discos magnéticos y almacenamiento flash

Los discos magnéticos proporcionan la mayor parte del almacenamiento secundario de los sistemas informáticos modernos. Aunque la capacidad de los discos ha crecido año tras año, los requisitos de almacenamiento de las grandes aplicaciones también han crecido muy rápido; en algunos casos, más que la capacidad de los discos. Una base de datos de gran tamaño puede necesitar centenares de discos. En los últimos años ha crecido rápidamente el almacenamiento en memorias flash, que se está convirtiendo en un competidor del almacenamiento en discos magnéticos para diversas aplicaciones.

10.2.1. Características físicas de los discos

Físicamente, los discos son relativamente sencillos (Figura 10.2). Cada **plato** del disco es de forma circular plana. Sus dos superficies están cubiertas por un material magnético en el que se graba la información. Los platos están hechos de metal rígido o de vidrio.

Mientras se utiliza el disco, un motor lo hace girar a una velocidad constante elevada (generalmente 60, 90 o 120 revoluciones por segundo, aunque existen discos que giran a 250 revoluciones por segundo). Una cabeza de lectura y escritura está colocada justo encima de la superficie del plato. La superficie del disco se divide a efectos lógicos en **pistas,** que se subdividen en **sectores.** Un **sector** es la unidad mínima de información que se puede leer o escribir en el disco. En los discos actuales, el tamaño de los sectores suele ser de 512 bytes; hay entre 50.000 y 100.000 pistas en cada plato y de uno a cinco platos por disco. Las pistas internas, las más cercanas al eje, son más cortas, y en los discos actuales, las pistas exteriores contienen más sectores que las internas; suele haber unos 500 a 1.000 sectores por pista en las pistas internas y alrededor de 1.000 a 2.000 en las externas. Estos números pueden variar de un modelo a otro; los discos de mayor capacidad tienen más sectores por pista y más pistas en cada plato.

La **cabeza de lectura y escritura** guarda la información en los sectores en forma de inversiones de la dirección de magnetización del material magnético.

Cada cara de un plato del disco posee una cabeza de lectura y escritura que se desplaza por el plato para tener acceso a las diferentes pistas. El disco suele contener muchos platos y las cabezas de lectura y escritura de todas las pistas están montadas en un único dispositivo denominado **brazo del disco** y se mueven conjuntamente. El conjunto de los platos del disco montados sobre un mismo eje y de las cabezas montadas en el brazo del disco se denomina **dispositivo cabeza-disco.** Dado que las cabezas de todos los platos se desplazan conjuntamente, cuando la cabeza se halle en la pista *i*-ésima de un plato, las restantes también se encontrarán en la pista *i*-ésima de sus platos respectivos. Por tanto, el conjunto de las pistas *i*-ésimas de todos los platos se denomina **cilindro** *i*-ésimo.

Actualmente dominan el mercado los discos con un diámetro de plato de tres pulgadas y media. Tienen un menor coste y tiempos de búsqueda más cortos, debido al menor tamaño, que los discos más grandes, de hasta catorce pulgadas, que eran habituales en el pasado y, aun así, ofrecen gran capacidad de almacenamiento. Se usan discos de diámetro incluso menor en dispositivos móviles como las computadoras portátiles, las de mano y los reproductores de música de bolsillo.

Las cabezas de lectura y escritura se mantienen lo más próximas posible a la superficie de los discos para aumentar la densidad de grabación. Las cabezas suelen flotar o volar tan solo a unas micras de la superficie de cada disco; el giro del disco crea una pequeña corriente de aire y el dispositivo de las cabezas se fabrica de manera que ese flujo de aire mantenga las cabezas flotando rasantes sobre la superficie de los discos. Como las cabezas flotan tan cercanas a la superficie, los platos se deben elaborar con esmero para que sean totalmente planos.

Los choques de las cabezas con la superficie de los platos pueden suponer un problema. Si la cabeza entra en contacto con la superficie del disco, puede arrancar el medio de grabación, lo que destruye los datos que allí hubiera. En los modelos antiguos, el contacto de la cabeza con la superficie hacía que el material arrancado flotase en el aire y se interpusiera entre las cabezas y los platos, lo que causaba más choques; por tanto, la caída de las cabezas podía dar lugar a un fallo de todo el disco. Los dispositivos actuales emplean como medio de grabación una fina capa de metal magnético. Son mucho menos susceptibles de fallar a causa del choque de las cabezas con las superficies de los platos que los antiguos discos recubiertos de óxido.

El **controlador de disco** actúa como interfaz entre el sistema informático y el hardware concreto de la unidad de disco; en los sistemas de disco modernos, el controlador se implementa dentro de la unidad de disco. El controlador de disco acepta los comandos de alto nivel de lectura o escritura en un sector dado e inicia las acciones correspondientes, como el desplazamiento del brazo del disco a la pista adecuada y la lectura o escritura real de los datos. Los controladores de disco también añaden **sumas de comprobación** a cada sector en el que se escribe; las sumas de comprobación se calculan a partir de los datos escritos en ese sector. Cuando se vuelve a realizar allí una operación de lectura, se vuelve a calcular esa suma a partir de los datos recuperados y se compara con el valor guardado; si los datos se han deteriorado, resulta muy probable que la suma de comprobación recién calculada no coincida con la guardada. Si se produce un error de este tipo, el controlador volverá a intentar la lectura varias veces; si el error persiste, el controlador indica un fallo de lectura.

Otra labor interesante llevada a cabo por los controladores de disco es la **reasignación de los sectores dañados.** Si el controlador detecta que un sector está dañado cuando se da formato al disco por primera vez, o cuando se realiza un intento de escribir allí, puede reasignar lógicamente el sector a una ubicación física diferente (escogida de entre un grupo de sectores adicionales preparados con esta finalidad). La reasignación se anota en disco o en memoria no volátil y la escritura se realiza en la nueva ubicación.

Los discos se conectan a los sistemas informáticos mediante conexiones de alta velocidad. Las interfaces más comunes para la conexión de los discos a las computadoras son: (1) **SATA,** que significa **serial ATA**[1] (ATA serie) y una nueva versión de SATA llamada SATA II o SATA 3 GB (las versiones más antiguas del estándar ATA, denominadas **PATA,** o Parallel ATA-ATA paralela, aún siguen en gran uso), (2) la **interfaz de conexión para sistemas informáticos pequeños** (SCSI: *small computer system interconnect*, pronunciado «escasi»), (3) SAS (que significa SCSI con conexión serie: *serial attached SCSI*) y (4) la interfaz *Fibre Channel*. Los sistemas de discos externos portátiles utilizan generalmente las interfaces USB o IEEE 1394 FireWire.

Aunque los discos se suelen conectar directamente a la interfaz de disco de la computadora mediante cables, también pueden hallarse en una ubicación remota y conectarse mediante una red de alta velocidad al controlador de disco. En la arquitectura de **red de área de almacenamiento** (*storage area network*: SAN) se conectan un gran número de discos a varias computadoras servidoras mediante una red de alta velocidad. Los discos suelen disponerse localmente mediante una técnica de organización de almacenamiento denominada **disposición redundante de discos independientes** (*redundant arrays of independent disks*: RAID) (que se describe en la Sección 10.3) para proporcionar a los servidores la vista lógica de un solo disco de gran tamaño y muy fiable. La computadora y el subsistema de disco siguen usando las interfaces SCSI, SAS o Fibre Channel para comunicarse entre sí, aunque puedan estar separados por una red. El acceso remoto a los discos también implica que pueden compartirse entre varias computadoras que pueden ejecutar en paralelo diferentes partes de una misma aplicación. El acceso remoto también supone que los discos que contienen datos importantes se pueden guardar en una sala en la que los administradores del sistema pueden supervisarlos y mantenerlos, en lugar de estar dispersos por diferentes partes de la organización.

El **almacenamiento conectado en red** (*network attached storage*: NAS) es una alternativa a SAN. NAS es muy parecido a SAN, salvo que, en lugar de que el almacenamiento en red aparezca como un gran disco, proporciona una interfaz al sistema de archivos que utiliza protocolos de sistemas de archivos en red como NFS o CIFS.

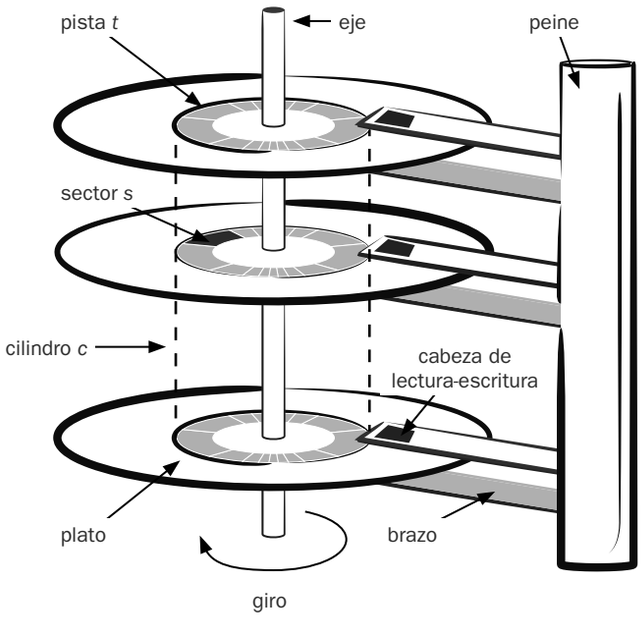


Figura 10.2. Mecanismo de disco de cabezas móviles.

1 ATA es una conexión para dispositivos de almacenamiento, normalizada desde los años ochenta.

10.2.2. Medidas del rendimiento de los discos

Las principales medidas de la calidad de los discos son su capacidad, su tiempo de acceso, su velocidad de transferencia de datos y su fiabilidad.

El **tiempo de acceso** es el tiempo transcurrido desde que se formula una solicitud de lectura o de escritura hasta que comienza la transferencia de datos. Para tener acceso (es decir, para leer o escribir) a los datos de un sector dado del disco, primero se debe desplazar el brazo para que se ubique sobre la pista correcta y luego hay que esperar a que el sector aparezca bajo él debido a la rotación del disco. El tiempo para volver a ubicar el brazo se denomina **tiempo de búsqueda** y aumenta con la distancia que deba recorrer. Los tiempos de búsqueda típicos varían de dos a treinta milisegundos, en función de la distancia de la pista a la posición inicial del brazo. Los discos de menor tamaño tienden a tener tiempos de búsqueda menores, dado que la cabeza tiene que recorrer una distancia inferior.

El **tiempo medio de búsqueda** es el promedio de los tiempos de búsqueda medido en una sucesión de solicitudes aleatorias (uniformemente distribuidas). Si todas las pistas tienen el mismo número de sectores y se desprecia el tiempo necesario para que la cabeza comience a moverse y se detenga, se puede demostrar que el tiempo medio de búsqueda es un tercio del peor de los tiempos de búsqueda posibles. Teniendo en cuenta estos factores, el tiempo medio de búsqueda es aproximadamente la mitad del tiempo máximo de búsqueda. Los tiempos medios de búsqueda varían actualmente entre cuatro y diez milisegundos, dependiendo del modelo de disco.

Una vez que la cabeza ha alcanzado la pista deseada, el tiempo de espera hasta que el sector al que hay que acceder aparece bajo la cabeza se denomina **tiempo de latencia rotacional.** Las velocidades rotacionales de los discos varían actualmente entre 5.400 rotaciones por minuto (90 por segundo) y 15.000 rotaciones por minuto (250 rotaciones por segundo) o, lo que es lo mismo, de cuatro a 11,1 milisegundos por rotación. En promedio, hace falta media rotación del disco para que aparezca bajo la cabeza el comienzo del sector deseado. Por tanto, el **tiempo de latencia medio** del disco es la mitad del tiempo empleado en una rotación completa del disco.

El tiempo de acceso es la suma del tiempo de búsqueda y del tiempo de latencia y varía de ocho a veinte milisegundos. Una vez situado bajo la cabeza el primer sector de los datos, comienza la transferencia. La **velocidad de transferencia de datos** es la velocidad a la que se pueden recuperar datos del disco o guardarlos en él. Los sistemas de disco actuales soportan velocidades máximas de transferencia de 25 a 100 megabytes por segundo; las velocidades de transferencia medias son significativamente menores que la velocidad de transferencia máxima para las pistas interiores del disco, dado que estas tienen menos sectores. Por ejemplo, un disco con una velocidad de transferencia máxima de 100 megabytes por segundo puede tener una velocidad de transferencia continuada de alrededor de 30 megabytes por segundo en sus pistas internas.

La última medida que se estudiará de entre las utilizadas habitualmente es el **tiempo medio entre fallos,** que indica una medida de la fiabilidad del disco. El tiempo medio entre fallos de un disco (o de cualquier otro sistema) es la cantidad de tiempo que, en promedio, se puede esperar que el sistema funcione de manera continua sin tener ningún fallo. De acuerdo con las afirmaciones de los fabricantes, el tiempo medio entre fallos de los discos actuales varía entre 500.000 y 1.200.000 horas (de 57 a 136 años). En la práctica, el tiempo medio entre fallos anunciado se calcula en términos de la probabilidad de fallo cuando el disco es nuevo (este dato significa que, dados 1.000 discos nuevos, si el tiempo medio entre fallos es de 1.200.000 horas, en promedio, uno de ellos fallará cada 1.200 horas). Un tiempo medio entre fallos de 1.200.000 horas no implica que se pueda esperar que el disco funcione 136 años. La mayoría de los discos tienen una esperanza de vida de unos cinco años, y tienen tasas de fallo significativamente más altas en cuanto alcanzan cierto tiempo.

Las unidades de disco para las máquinas de sobremesa incorporan normalmente la interfaz SATA (Serial ATA), la cual proporciona velocidades de transferencia de 150 megabytes por segundo, o la interfaz SATA II 3 GB, capaz de soportar 300 megabytes por segundo. La interfaz PATA 5 soportaba velocidades de transferencia de 133 megabytes por segundo. Las unidades de disco diseñadas para los sistemas servidores suelen soportar las interfaces Ultra320 SCSI, que permite hasta 320 megabytes por segundo, y Conexión serie SCSI, que proporciona tasas de transferencia de 3 a 6 gigabits por segundo. Los dispositivos SAN de red de área de almacenamiento que se conectan a los servidores por una red, normalmente usan Fibre Channel FC 2-GB o 4-GB, que proporciona tasas de hasta 256 o 512 megabytes por segundo. La velocidad de transferencia de cada interfaz se comparte entre todos los discos conectados a la misma, excepto en las interfaces serie, que solo permiten que se conecte un disco a cada interfaz.

10.2.3. Optimización del acceso a los bloques del disco

Las solicitudes de E/S al disco las generan tanto el sistema de archivos como el gestor de la memoria virtual presente en la mayor parte de los sistemas operativos. Cada solicitud especifica la dirección del disco a la que hay que hacer referencia; esa dirección se expresa como un *número de bloque*. Un **bloque** es una unidad lógica que consiste en un número fijo de sectores contiguos. El tamaño de los bloques varía de 512 bytes a varios kilobytes. Entre el disco y la memoria principal los datos se transfieren por bloques. El término **página** también se suele usar para los bloques, aunque en algunos pocos contextos, como las memorias flash, se refieren a cosas diferentes.

La búsqueda de bloques en una petición secuencial se puede clasificar como un patrón secuencial o aleatorio. En un patrón de **acceso secuencial,** las peticiones sucesivas se refieren a números de bloques consecutivos, que se encuentran en la misma pista o en pistas adyacentes. Para leer un bloque en un acceso secuencial, primero se requiere un tiempo de búsqueda, pero las peticiones sucesivas no requieren tiempo de búsqueda o solo a la pista adyacente, que es más rápido que a una pista que se encuentre más lejos.

Al contrario, en un patrón de **acceso aleatorio,** las sucesivas peticiones de bloques se encuentran localizadas aleatoriamente en el disco. Cada petición requiere un tiempo de búsqueda. El número de accesos a bloques aleatorios que puede satisfacer un único disco en un segundo depende del tiempo de búsqueda y suele ser de unos 100 a 200 accesos por segundo. Como solo se lee una pequeña cantidad de datos (un bloque) por cada tiempo de búsqueda, la tasa de transferencia es significativamente más lenta con un patrón aleatorio que con uno secuencial.

Se han desarrollado un conjunto de técnicas que permiten mejorar la velocidad de acceso a los bloques.

- **Memoria intermedia («buffer»).** Los bloques que se leen del disco se almacenan temporalmente en una memoria intermedia llamada buffer, para satisfacer futuras peticiones. Este sistema lo emplea tanto el sistema operativo como el sistema de bases de datos. El uso de memoria intermedia por las bases de datos se trata con más detalle en la Sección 10.8.
- **Lectura adelantada.** Cuando se accede a un bloque de un disco se leen en una memoria intermedia más bloques consecutivos de la misma pista incluso aunque no haya peticiones de lectura de bloques pendientes. En el caso de que se produzca un acceso se-

cuencial, esta lectura adelantada asegura que ya se encuentran en memoria muchos bloques cuando se pidan, y se minimiza el tiempo que se desperdicia en los tiempos de búsqueda y la latencia rotacional por cada lectura de un bloque. Los sistemas operativos también realizan lectura adelantada de bloques consecutivos del sistema de archivos. La lectura adelantada, sin embargo, no es muy útil en el caso de accesos de bloques aleatorios.

- **Planificación.** Si hay que transferir varios bloques de un mismo cilindro desde el disco a la memoria principal es posible disminuir el tiempo de acceso solicitando los bloques en el orden en el que pasarán por debajo de las cabezas. Si los bloques deseados se hallan en cilindros diferentes resulta ventajoso solicitar los bloques en un orden que minimice el movimiento del brazo del disco. Los algoritmos de **planificación del brazo del disco** intentan ordenar el acceso a las pistas de manera que se aumente el número de accesos que se puedan procesar. Un algoritmo utilizado con frecuencia es el **algoritmo del ascensor,** que funciona de manera parecida a muchos ascensores. Suponga que, inicialmente, el brazo se desplaza desde la pista más interna hacia el exterior del disco. Bajo el control del algoritmo del ascensor, el brazo se detiene en cada pista para la que haya una solicitud de acceso, atiende las peticiones para esa pista y continúa desplazándose hacia el exterior hasta que no queden solicitudes pendientes para pistas más externas. En ese punto, el brazo cambia de dirección, se desplaza hacia el interior y vuelve a detenerse en cada pista solicitada hasta que alcanza una pista en la que no haya solicitudes para pistas más cercanas al centro del disco. Entonces cambia de dirección e inicia un nuevo ciclo. Los controladores de disco suelen realizar la labor de reordenar las solicitudes de lectura para mejorar el rendimiento, dado que conocen perfectamente la organización de los bloques del disco, la posición rotacional de los platos y la posición del brazo.
- **Organización de archivos.** Para reducir el tiempo de acceso a los bloques se pueden organizar los bloques del disco de una manera que se corresponda fielmente con la forma en que se espera tener acceso a los datos. Por ejemplo, si se espera tener acceso secuencial a un archivo, se deben guardar secuencialmente en cilindros adyacentes todos los bloques del archivo. Los sistemas operativos más antiguos, como los de IBM para grandes sistemas, ofrecían a los programadores un control detallado de la ubicación de los archivos, lo que permitía reservar un conjunto de cilindros para guardar un archivo. Sin embargo, este control supone una carga para el programador o el administrador del sistema porque tienen que decidir, por ejemplo, los cilindros que debe asignar a cada archivo. Esto puede exigir una costosa reorganización si se insertan datos en el archivo o se borran de él.

  Los sistemas operativos posteriores, como Unix y Microsoft Windows, ocultan a los usuarios la organización del disco y gestionan la asignación de manera interna. Aunque no garantizan que todos los bloques de un archivo se asignen secuencialmente, asignan varios bloques consecutivos (una **extensión**) de una vez para un archivo. Entonces un acceso secuencial al archivo solo necesita un tiempo de búsqueda por extensión, en lugar de un tiempo de búsqueda por bloque. Sin embargo, con el transcurso del tiempo, un archivo secuencial que tiene muchas adiciones puede quedar **fragmentado;** es decir, sus bloques pueden quedar dispersos por el disco. Para reducir la fragmentación, el sistema puede hacer una copia de seguridad de los datos del disco y restaurarlo completamente. La operación de restauración vuelve a escribir de manera contigua (o casi) los bloques de cada archivo. Algunos sistemas (como las diferentes versiones del sistema operativo Windows) disponen de utilidades que examinan el disco y desplazan los bloques para reducir la fragmentación. Las mejoras de rendimiento obtenidas en algunos casos con estas técnicas son elevadas.
- **Memoria de escritura no volátil.** Dado que el contenido de la memoria se pierde durante los fallos de suministro eléctrico, hay que guardar en disco la información sobre las actualizaciones de las bases de datos para que superen las posibles caídas del sistema. Por esta razón, el rendimiento de las aplicaciones de bases de datos con gran cantidad de actualizaciones, como los sistemas de procesamiento de transacciones, depende mucho de la velocidad de escritura en el disco.

  Se puede utilizar **memoria no volátil de acceso aleatorio** (RAM no volátil o NV-RAM: *nonvolatile random-access memory*) para acelerar drásticamente la escritura en el disco. El contenido de la NV-RAM no se pierde durante los fallos del suministro eléctrico. Una manera habitual de implementar la NV-RAM es utilizar RAM alimentada por baterías, aunque cada vez más se usan memorias flash. La idea es que, cuando el sistema de bases de datos (o el sistema operativo) solicita que se escriba un bloque en el disco, el controlador del disco escriba el bloque en una memoria intermedia de NV-RAM y comunique de manera inmediata al sistema operativo que la escritura se completó con éxito. El controlador escribirá los datos en el disco cuando no haya otras solicitudes o cuando se llene la memoria intermedia de NV-RAM. Cuando el sistema de bases de datos solicita la escritura de un bloque, solo percibe un retraso si la memoria intermedia de NV-RAM se encuentra llena. Durante la recuperación de una caída del sistema se vuelven a escribir en el disco todas las operaciones de escritura pendientes en la memoria intermedia de NV-RAM. La memoria NV-RAM se encuentra en ciertos discos de altas prestaciones, pero es más frecuente encontrarlo en «controladores RAID»; se tratará sobre RAID en la Sección 10.3.
- **Disco de registro histórico.** Otra manera de reducir las latencias de escritura es utilizar un disco de registro histórico (es decir, un disco destinado a escribir un registro secuencial) de manera muy parecida a la memoria intermedia RAM no volátil. Todos los accesos al disco de registro histórico son secuenciales, lo que básicamente elimina el tiempo de búsqueda y, así, se pueden escribir simultáneamente varios bloques consecutivos, lo que hace que los procesos de escritura en el disco de registro sean varias veces más rápidos que los procesos de escritura aleatorios. Igual que ocurría anteriormente, también hay que escribir los datos en su ubicación verdadera en el disco, pero el disco de registro puede realizar este proceso de escritura más adelante, sin que el sistema de bases de datos tenga que esperar a que se complete. Además, el disco de registro histórico puede reordenar las operaciones de escritura para reducir el movimiento del brazo. Si el sistema se cae antes de que se haya completado alguna operación de escritura en la ubicación real del disco, cuando el sistema se recupera, lee el disco de registro histórico para averiguar las operaciones de escritura que no se han completado y las lleva a cabo.

  Los sistemas de archivo que soportan los discos de registro histórico mencionados se denominan **sistemas de archivos de diario.** Los sistemas de archivos de diario se pueden implementar incluso sin un disco de registro histórico independiente, guardando los datos y el registro histórico en el mismo disco. Al hacerlo así se reduce el coste económico, a expensas de un menor rendimiento.

  La mayoría de los sistemas de archivos modernos implementan un diario y usan el disco de registro histórico al escribir información interna del sistema de archivos, como la relativa a la asignación de archivos. Los primeros sistemas de archivos permitían reordenar las operaciones de escritura sin emplear disco de registro, y corrían el riesgo de que las estructuras de datos del sistema de archivos del disco se corrompiesen si se producía una caída del sistema. Suponga, por ejemplo, que un

sistema de archivos utiliza una lista enlazada y se inserta un nuevo nodo al final escribiendo primero los datos del nuevo nodo y actualizando posteriormente el puntero del nodo anterior. Suponga también que las operaciones de escritura se reordenasen, de forma que se actualizase primero el puntero y que el sistema se cayese antes de escribir el nuevo nodo. El contenido del nodo sería la basura que hubiese anteriormente en el disco, lo que daría lugar a una estructura de datos corrupta.

Para evitar esta posibilidad, los primeros sistemas de archivo tenían que realizar una comprobación de consistencia del sistema al reiniciar este para asegurarse de que las estructuras de datos eran consistentes. Si no lo eran, había que tomar medidas adicionales para volver a hacerlas consistentes. Estas comprobaciones daban lugar a grandes retardos en el inicio del sistema después de las caídas, que se agravaron al aumentar la capacidad de las unidades de disco. Los sistemas de archivo de diario permiten un reinicio rápido sin necesidad de esas comprobaciones.

No obstante, las operaciones de escritura realizadas por las aplicaciones no se suelen escribir en el disco del registro histórico. Los sistemas de bases de datos implementan sus propias variantes de registro, que se estudian posteriormente en el Capítulo 16.

10.2.4. Almacenamiento flash

Como se ha mencionado en la Sección 10.1, existen dos tipos de memoria flash, NOR y NAND. La memoria flash NOR permite el acceso aleatorio a palabras individuales de la memoria y tiene un tiempo de lectura comparable al de la memoria principal. Sin embargo, al contrario que las memorias flash NOT, la lectura de memorias flash NAND requiere la lectura de una *página* completa de datos, normalmente entre 512 y 4.096 bytes, que se llevan de la memoria flash NAND a la memoria principal. Las páginas de la memoria flash NAND son similares a los sectores de los discos magnéticos. Pero la memoria flash NAND es significativamente más barata que las memorias flash NOR y tiene mayor capacidad de almacenamiento siendo, con mucho, la más usada.

Los sistemas de almacenamiento que utilizan memorias flash NAND proporcionan la misma interfaz orientada a bloques que los sistemas de disco. Comparadas con los discos magnéticos, las memorias flash proporcionan un acceso más rápido: una página de datos se puede leer en 1 o 2 microsegundos, comparado con un acceso aleatorio en un disco, que puede tardar de 5 a 10 milisegundos. Las memorias flash tienen menor tasa de transferencia que los discos magnéticos, siendo común unos 20 megabytes por segundo. Algunas memorias flash recientes han incrementado su tasa de transferencia hasta los 100 o 200 megabytes por segundo. Sin embargo, las unidades de disco de estado sólido utilizan varios chips de memoria flash en paralelo, para aumentar las tasas de transferencia hasta los 200 megabytes por segundo, que es mayor que en la mayoría de los discos magnéticos.

Las escrituras en una memoria flash son un poco más complicadas. Una escritura de una página suele tardar unos pocos microsegundos. Sin embargo, una vez escrita la página no se puede sobrescribir directamente. En su lugar, hay que borrarla y después sobrescribirla. La operación de borrado se puede realizar en un número de páginas llamadas **bloque de borrado,** y tarda de 1 a 2 milisegundos. El tamaño de los bloques de borrado, a menudo llamados simplemente «bloques» en la literatura de memorias flash, suele ser significativamente mayor que el tamaño de los bloques en los sistemas de almacenamiento. Más aún, existe un límite en el número de veces que se puede borrar una página flash, normalmente entre 100.000 y 1.000.000 de veces. Cuando se alcanza este límite, es muy probable que se produzcan errores en el almacenamiento de bits.

Los sistemas de memorias flash limitan el impacto tanto de los borrados lentos como de los límites de actualización, haciendo una correspondencia entre números de página lógicos a números de página físicos. Cuando se actualiza una página lógica, se puede reasignar a cualquier página física ya borrada y la página original se borrará más adelante. Cada página física tiene una pequeña área de memoria en la que se almacena su dirección lógica; si la dirección lógica se reasigna a una página física diferente, la página física original se marca como borrada. Por tanto, revisando las páginas físicas se puede encontrar dónde se encuentra cada página lógica. La correspondencia entre páginas lógicas y físicas se replica en una **tabla de traducción** en memoria para un rápido acceso.

Los bloques que contienen varias páginas borradas se borran periódicamente teniendo cuidado primero de copiar las páginas no borradas a otros bloques diferentes (la tabla de traducción se actualiza para las páginas no borradas). Como las páginas físicas se pueden actualizar solo un número fijo de veces, las páginas físicas que se han borrado muchas veces se marcan como «datos fríos», es decir, datos que se actualizan muy raramente, mientras que las páginas que no se han borrado tantas veces se usan para almacenar «datos calientes», es decir, los datos que se actualizan con frecuencia. Este principio de distribuir las operaciones de borrado entre los bloques físicos se denomina **equilibrado del uso** (*wear leveling*) y lo suelen realizar de forma transparente los controladores de las memorias flash. Si una página física se daña por un número excesivo de actualizaciones, se puede eliminar de uso, sin afectar a la memoria flash completa.

Todas las acciones anteriores se llevan a cabo en una capa de software llamada **capa de traducción flash;** por encima de esta capa el almacenamiento en memoria flash parece idéntico al de un disco magnético, proporcionando la misma interfaz orientada a página/sector, excepto que el almacenamiento flash es más rápido. Las estructuras de los sistemas de archivo y almacenamiento de bases de datos tendrán una vista lógica idéntica de la estructura de almacenamiento subyacente, independientemente de si el almacenamiento se produce en una memoria magnética o flash.

Las **unidades de disco híbridas** son sistemas de discos duros que combinan almacenamiento magnético con una pequeña cantidad de memoria flash, que se utilizan como caché para los datos a los que se accede con frecuencia. Los datos que se usan con frecuencia y no se suelen actualizar son ideales para mantenerlos en la memoria flash.

10.3. RAID

Los requisitos de almacenamiento de datos de algunas aplicaciones (en particular las aplicaciones web, las de bases de datos y las multimedia) han crecido tan rápidamente que hace falta un gran número de discos para almacenar sus datos, pese a que la capacidad de los discos también ha aumentado mucho.

Tener gran número de discos en el sistema proporciona oportunidades para mejorar la velocidad a la que se pueden leer o escribir los datos si los discos funcionan en paralelo. También se pueden realizar varias operaciones de lectura o escritura independientes simultáneamente. Además, esta configuración ofrece la posibilidad de mejorar la fiabilidad del almacenamiento de datos, ya que se puede guardar información repetida en varios discos. Por tanto, el fallo de uno de los discos no provoca pérdidas de datos.

Para conseguir mayor rendimiento y fiabilidad se han propuesto varias técnicas de organización de los discos, denominadas colectivamente **disposición redundante de discos independientes** (RAID: *redundant array of independent disks*).

En el pasado, los diseñadores de sistemas consideraron los sistemas de almacenamiento compuestos de varios discos pequeños de bajo coste como alternativa económicamente efectiva al empleo de unidades grandes y caras; el coste por megabyte de los discos más pequeños era menor que el de los de gran tamaño. De hecho, la I de RAID, que ahora significa *independientes*, originalmente representaba *económicos* (*inexpensive*). Hoy en día, sin embargo, todos los discos son de pequeño tamaño y los discos de mayor capacidad tienen un menor coste por megabyte. Los sistemas RAID se utilizan por su mayor fiabilidad y por su mejor rendimiento, más que por motivos económicos. Otro motivo fundamental del empleo de RAID es su mayor facilidad de administración y de operación.

10.3.1. Mejora de la fiabilidad mediante la redundancia

Considere en primer lugar la fiabilidad. La probabilidad de que falle, al menos, un disco de un conjunto de N discos es mucho más elevada que la posibilidad de que falle un único disco concreto. Supóngase que el tiempo medio entre fallos de un disco dado es de 100.000 horas o, aproximadamente, once años. Por tanto, el tiempo medio entre fallos de algún disco de una disposición de cien discos será de $100.000/100 = 1.000$ horas, o cuarenta y dos días, ¡lo que no es mucho! Si solo se guarda una copia de los datos, cada fallo de disco dará lugar a la pérdida de una cantidad significativa de datos (como ya se estudió en la Sección 10.2.1). Una frecuencia de pérdida de datos tan elevada resulta inaceptable.

La solución al problema de la fiabilidad es introducir la **redundancia;** es decir, se guarda información adicional que normalmente no se necesita pero que se puede utilizar en caso de fallo de un disco para reconstruir la información perdida. Por tanto, aunque falle un disco, no se pierden datos, por lo que el tiempo medio efectivo entre fallos aumenta, siempre que se cuenten solo los fallos que dan lugar a una pérdida de datos o a su no disponibilidad.

El enfoque más sencillo (pero el más costoso) para la introducción de la redundancia es duplicar todos los discos. Esta técnica se denomina **creación de imágenes** (o, a veces, *creación de sombras*). Cada unidad lógica consta, por tanto, de dos unidades físicas y cada proceso de escritura se lleva a cabo por duplicado. Si uno de los discos falla se pueden leer los datos del otro. Los datos solo se perderán si falla el segundo disco antes de que se repare el primero que falló.

El tiempo medio entre fallos (entendiendo fallo como pérdida de datos) de un disco con imagen depende del tiempo medio entre fallos de cada disco y del **tiempo medio de reparación,** que es el tiempo que se tarda (en promedio) en sustituir un disco averiado y en restaurar sus datos. Supóngase que los fallos de los dos discos son *independientes*; es decir, no hay conexión entre el fallo de uno y el del otro. Por tanto, si el tiempo medio entre fallos de un solo disco es de cien mil horas y el tiempo medio de reparación es de diez horas, el **tiempo medio entre pérdidas de datos** de un sistema de discos con imagen es $100.000^2/(2 * 10) = 500 * 10^6$ horas, o ¡57.000 años! (aquí no se entra en detalle sobre los cálculos; se proporcionan en las referencias de las notas bibliográficas).

Hay que tener en cuenta que la suposición de independencia de los fallos de los discos no resulta válida. Los fallos en el suministro eléctrico y los desastres naturales como los terremotos, los incendios y las inundaciones pueden dar lugar a daños simultáneos de ambos discos. A medida que los discos envejecen, la probabilidad de fallo aumenta, lo que incrementa la posibilidad de que falle el segundo disco mientras se repara el primero. A pesar de todas estas consideraciones, los sistemas de discos con imagen ofrecen una fiabilidad mucho más elevada que los sistemas de disco único. Hoy en día se dispone de sistemas de discos con imagen con un tiempo medio entre pérdidas de datos de entre 500.000 y 1.000.000 de horas, de 55 a 110 años.

Los fallos en el suministro eléctrico son una causa especial de preocupación, dado que tienen lugar con mucha más frecuencia que los desastres naturales. Los fallos en el suministro eléctrico no son un problema si no se está realizando ninguna transferencia de datos al disco cuando estos se producen. Sin embargo, incluso con la creación de imágenes de los discos, si se hallan en curso procesos de escritura en el mismo bloque en ambos discos y el suministro eléctrico falla antes de que se haya acabado de escribir en ambos bloques, los dos pueden quedar en un estado inconsistente. La solución a este problema es escribir primero una copia y luego la otra, de modo que una de las copias siempre sea consistente. Cuando se vuelve a iniciar el sistema tras un fallo en el suministro eléctrico hacen falta algunas acciones adicionales para recuperar los procesos de escritura incompletos. Este asunto se examina en el Ejercicio práctico 10.3.

10.3.2. Mejora del rendimiento mediante el paralelismo

Considere ahora las ventajas del acceso en paralelo a varios discos. Con la creación de imágenes de los discos, la velocidad a la que se pueden procesar las solicitudes de lectura se duplica, dado que pueden enviarse a cualquiera de los discos (siempre y cuando los dos integrantes de la pareja estén operativos, como es el caso habitual). La velocidad de transferencia de cada proceso de lectura es la misma que en los sistemas de disco único, pero el número de procesos de lectura por unidad de tiempo se ha duplicado.

También se puede mejorar la velocidad de transferencia con varios discos **distribuyendo los datos** entre ellos. En su forma más sencilla, la distribución de datos consiste en dividir los bits de cada byte entre varios discos; esto se denomina **distribución en el nivel de bits.** Por ejemplo, si se dispone de una disposición de ocho discos, se puede escribir el bit i de cada byte en el disco i. La disposición de ocho discos puede tratarse como un solo disco con sectores que tienen ocho veces el tamaño normal y, lo que es más importante, que tienen ocho veces la velocidad de acceso habitual. En una organización así cada disco toma parte en todos los accesos (de lectura o de escritura), por lo que el número de accesos que pueden procesarse por segundo es aproximadamente el mismo que con un solo disco, pero cada acceso puede octuplicar el número de datos leídos por unidad de tiempo respecto a un solo disco. La distribución en el nivel de bits puede generalizarse a cualquier número de discos que sea múltiplo o divisor de ocho. Por ejemplo, si se utiliza una disposición de cuatro discos, los bits i y $4 + i$ de cada byte irán al disco i.

La **distribución en el nivel de bloques** reparte cada bloque entre varios discos. Trata la disposición de discos como un único disco de gran capacidad y asigna números lógicos a los bloques; se da por supuesto que los números de bloque comienzan en cero. Con una disposición de n discos, la distribución en el nivel de bloques asigna el bloque lógico i de la disposición de discos al disco (i mod n) $+ 1$; utiliza el bloque físico $[i/n]$-ésimo del disco para almacenar el bloque lógico i. Por ejemplo, con ocho discos, el bloque lógico 0 se almacena en el bloque físico 0 del disco 1, mientras que el bloque lógico 11 se almacena en el bloque físico 1 del disco 4. Si se lee un archivo grande, la distribución en el nivel de bloques busca simultáneamente n bloques en paralelo en los n discos, lo que permite una gran velocidad de transferencia para lecturas de gran tamaño. Cuando se lee un solo bloque, la velocidad de transferencia de datos es igual que con un único disco, pero los restantes $n - 1$ discos quedan libres para realizar otras acciones.

La distribución en el nivel de bloques es la forma de distribución de datos más usada. También son posibles otros niveles de distribución, como el de los bytes de cada sector o el de los sectores de cada bloque.

En resumen, en un sistema de discos, el paralelismo tiene dos objetivos principales:

1. Equilibrar la carga de varios accesos de pequeño tamaño (accesos a bloques), de manera que aumente la productividad de ese tipo de accesos.
2. Convertir en paralelos los accesos de gran tamaño para que su tiempo de respuesta se reduzca.

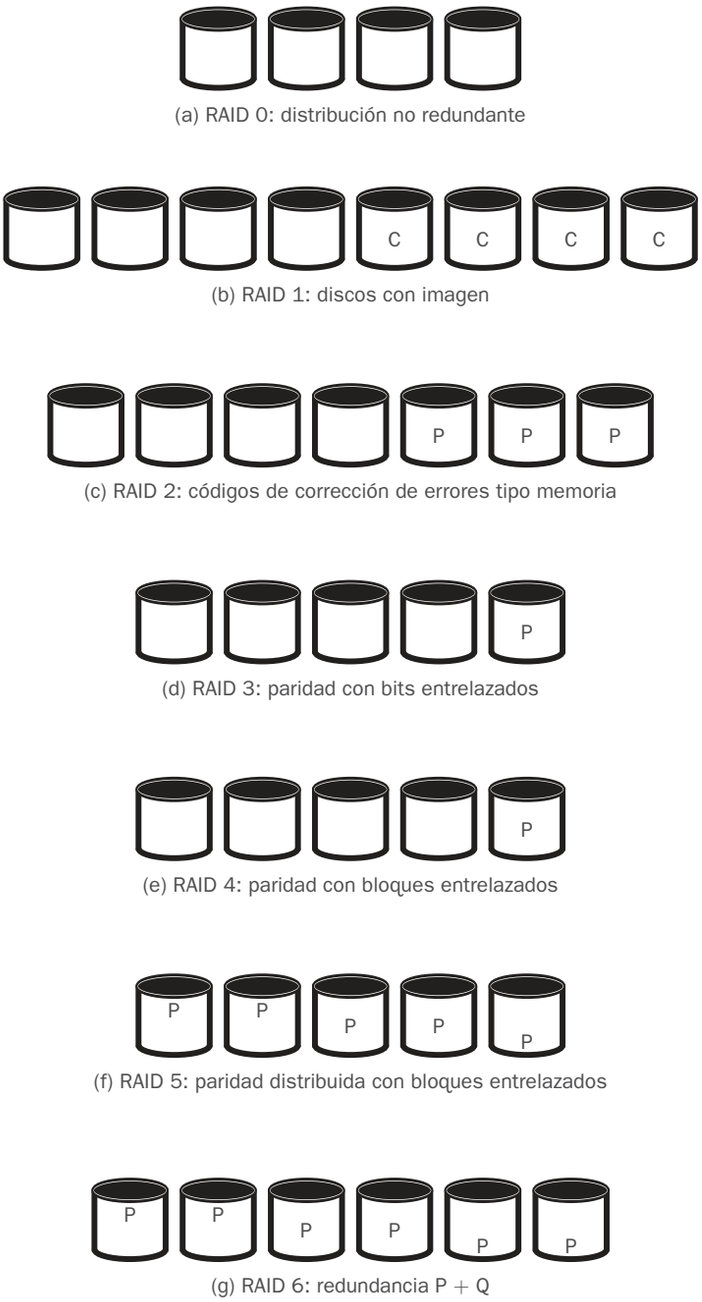


Figura 10.3. Niveles de RAID.

10.3.3. Niveles de RAID

La creación de imágenes proporciona gran fiabilidad pero resulta costosa. La distribución proporciona velocidades de transferencia de datos elevadas, pero no mejora la fiabilidad. Varios esquemas alternativos pretenden proporcionar redundancia a un coste menor combinando la distribución de los discos con los bits de «paridad» (que se describen a continuación). Estos esquemas tienen diferentes compromisos entre el coste y el rendimiento y se clasifican en los denominados **niveles de RAID,** como se muestra en la Figura 10.3 (en la figura, P indica los bits para la corrección de errores y C indica una segunda copia de los datos). La figura muestra cuatro discos de datos para todos los niveles, y los discos adicionales se utilizan para guardar la información redundante para la recuperación en caso de fallo.

- **Nivel 0 de RAID** hace referencia a disposiciones de discos con distribución en el nivel de bloques, pero sin redundancia (como la creación de imágenes o los bits de paridad). La Figura 10.3a muestra una disposición de tamaño cuatro.
- **Nivel 1 de RAID** hace referencia a la creación de imágenes del disco con distribución de bloques. La Figura 10.3b muestra una organización con imagen que guarda cuatro discos de datos.

  Observe que algunos fabricantes emplean el término **nivel 1+0 de RAID** o **nivel 10 de RAID** para referirse a imágenes del disco con distribución, y utilizan el término nivel 1 de RAID para las imágenes del disco sin distribución. Esto último también se puede emplear con disposiciones de discos para dar la apariencia de un disco grande y fiable: si cada disco tiene M bloques, los bloques lógicos 0 a M-1 se almacenan en el disco 0; M a $2M$-1 en el disco 1 (el segundo disco), y así sucesivamente, y se crea una imagen de cada disco.[2]
- **Nivel 2 de RAID,** también conocido como organización de códigos de corrección de errores tipo memoria (memory-style-error-correcting-code organization, ECC), emplea bits de paridad. Hace tiempo que los sistemas de memoria utilizan los bits de paridad para la detección y corrección de errores. Cada byte del sistema de memoria puede tener asociado un bit de paridad que registra si el número de bits del byte que valen uno es par (paridad = 0) o impar (paridad = 1). Si alguno de los bits del byte se deteriora (un uno se transforma en cero y viceversa) la paridad del byte se modifica y, por tanto, no coincide con la paridad guardada. Análogamente, si el bit de paridad guardado se deteriora no coincide con la paridad calculada. Por tanto, el sistema de memoria detecta todos los errores que afectan a un número impar de bits de un mismo byte. Los esquemas de corrección de errores guardan dos o más bits adicionales y pueden reconstruir los datos si se deteriora un solo bit.

  La idea de los códigos para la corrección de errores se puede utilizar directamente en las disposiciones de discos mediante la distribución de los bytes entre los diferentes discos. Por ejemplo, el primer bit de cada byte puede guardarse en el disco 0, el segundo en el disco 1, etc., hasta que se guarde el octavo bit en el disco 7; y los bits para la corrección de errores se guardan en discos adicionales.

  La Figura 10.3c muestra el esquema de nivel 2. Los discos marcados como P guardan los bits para la corrección de errores. Si uno de los discos falla, los restantes bits del byte y los de corrección de errores asociados se pueden leer en los demás discos y se pueden utilizar para reconstruir los datos deteriorados. La Figura 10.3c muestra una disposición de tamaño cuatro; observe que el RAID de nivel 2 solo necesita tres discos adicionales para los cuatro discos de datos, a diferencia del RAID de nivel 1, que necesitaba cuatro discos adicionales.

2 Observe que algunos fabricantes usan el término RAID 0+1 para referirse a una versión de RAID que utiliza la distribución para crear una disposición RAID 0, y crea una imagen de esa disposición en otra disposición, con la diferencia respecto de RAID 1 de que, si falla un disco, la disposición RAID 0 que contiene el disco queda inutilizable. La disposición con imagen se puede seguir utilizando, así que no hay pérdida de datos. Esta organización es inferior a RAID 1 cuando falla un disco, ya que los demás discos de la disposición RAID 0 se pueden seguir usando en RAID 1, pero quedan inactivos en RAID 0+1.

- **Nivel 3 de RAID,** organización de paridad con bits entrelazados, mejora respecto al nivel 2 al aprovechar que los controladores de disco, a diferencia de los sistemas de memoria, pueden detectar si los sectores se han leído correctamente, por lo que se puede utilizar un solo bit de paridad para la corrección y detección de los errores. La idea es la siguiente: si uno de los sectores se deteriora, se sabe exactamente el sector que es y, para cada uno de sus bits, se puede determinar si es un uno o un cero calculando la paridad de los bits correspondientes de los sectores de los demás discos. Si la paridad de los bits restantes es igual que la paridad guardada, el bit perdido es un cero; en caso contrario, es un uno.

  El nivel 3 de RAID es tan bueno como el nivel 2, pero resulta menos costoso en cuanto al número de discos adicionales (solo tiene un disco adicional), por lo que el nivel 2 no se utiliza en la práctica. La Figura 10.3d muestra el esquema de nivel 3.

  El nivel 3 de RAID tiene dos ventajas respecto al nivel 1. Solo necesita un disco de paridad para varios discos normales, mientras que el nivel 1 necesita un disco imagen por cada disco, por lo que el nivel 3 reduce la necesidad de almacenamiento adicional. Dado que los procesos de lectura y de escritura de cada byte se distribuyen por varios discos, con la distribución de los datos en N fracciones la velocidad de transferencia para la lectura o la escritura de un solo bloque multiplica por N la de una organización RAID de nivel 1 que emplee la distribución entre N discos. Por otro lado, el nivel 3 de RAID permite un menor número de operaciones de E/S por segundo, ya que todos los discos tienen que participar en cada solicitud de E/S.
- **Nivel 4 de RAID,** organización de paridad con bloques entrelazados, emplea la distribución del nivel de bloques, como RAID 0, y, además, guarda un bloque de paridad en un disco independiente para los bloques correspondientes de los otros N discos. Este esquema se muestra gráficamente en la Figura 10.3e. Si falla uno de los discos, se puede utilizar el bloque de paridad con los bloques correspondientes de los demás discos para restaurar los bloques del disco averiado.

  La lectura de un bloque solo accede a un disco, lo que permite que los demás discos procesen otras solicitudes. Por tanto, la velocidad de transferencia de datos de cada acceso es menor, pero se pueden ejecutar en paralelo varios accesos de lectura, lo que produce una mayor velocidad global de E/S. Las velocidades de transferencia para los procesos de lectura de gran tamaño son elevadas, dado que se pueden leer todos los discos en paralelo; los procesos de escritura de gran tamaño también tienen velocidades de transferencia elevadas, dado que los datos y la paridad pueden escribirse en paralelo.

  Los procesos de escritura independientes de pequeño tamaño, por otro lado, no pueden realizarse en paralelo. La operación de escritura de un bloque tiene que acceder al disco en el que se guarda ese bloque, así como al disco de paridad, ya que hay que actualizar el bloque de paridad. Además, para calcular la nueva paridad hay que leer tanto el valor anterior del bloque de paridad como el del bloque que se escribe. Por tanto, un solo proceso de escritura necesita cuatro accesos a disco: dos para leer los dos bloques antiguos y otros dos para escribir los dos nuevos.
- **Nivel 5 de RAID,** paridad distribuida con bloques entrelazados, mejora respecto al nivel 4 al dividir los datos y la paridad entre todos los $N + 1$ discos, en vez de guardar los datos en N discos y la paridad en uno solo. En el nivel 5 todos los discos pueden atender a las solicitudes de lectura, a diferencia del nivel 4 de RAID, en el que el disco de paridad no puede participar, por lo que el nivel 5 aumenta el número total de solicitudes que pueden atenderse en una cantidad de tiempo dada. En cada conjunto de N bloques lógicos, uno de los discos guarda la paridad y los otros N guardan los bloques.

  La Figura 10.3f muestra esta configuración, en la que las P están distribuidas entre todos los discos. Por ejemplo, con una disposición de cinco discos, el bloque de paridad, marcado como Pk, para los bloques lógicos $4k$, $4k + 1$, $4k + 2$, $4k + 3$ se guarda en el disco (k mod 5); los bloques correspondientes de los otros cuatro discos guardan los cuatro bloques de datos $4k$ al $4k + 3$. La siguiente tabla indica la manera en que se disponen los primeros veinte bloques, numerados del 0 al 19, y sus bloques de paridad. El patrón mostrado se repite para los siguientes bloques.

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

  Observe que un bloque de paridad no puede guardar la paridad de los bloques del mismo disco, ya que un fallo del disco supondría la pérdida de los datos y de la paridad y, por tanto, no sería recuperable. El nivel 5 incluye al nivel 4, dado que ofrece mejor rendimiento de lectura y de escritura por el mismo coste, por lo que el nivel 4 no se utiliza en la práctica.
- **Nivel 6 de RAID,** denominado esquema de redundancia P+Q, es muy parecido al nivel 5, pero guarda información redundante adicional para la protección contra los fallos de disco múltiples. En lugar de utilizar la paridad, el nivel 6 utiliza códigos para la corrección de errores, como los códigos Reed-Solomon (véanse las notas bibliográficas). En el esquema de la Figura 10.3g se guardan dos bits de datos redundantes por cada cuatro bits de datos (a diferencia del único bit de paridad del nivel 5) y el sistema puede tolerar dos fallos del disco.

Finalmente, se debe destacar que se han propuesto varias modificaciones a los esquemas RAID básicos aquí descritos, y que los diferentes fabricantes emplean su propia terminología para las variantes.

10.3.4. Elección del nivel RAID

Los factores que se deben tener en cuenta para elegir el nivel RAID son:

- El coste económico de los requisitos adicionales de almacenamiento en disco.
- Los requisitos de rendimiento en términos de número de operaciones de E/S.
- El rendimiento cuando falla un disco.
- El rendimiento durante la reconstrucción (esto es, mientras los datos del disco averiado se reconstruyen en un disco nuevo).

El tiempo que se tarda en reconstruir los datos que contenía el disco averiado puede ser significativo, y varía con el nivel RAID utilizado. La reconstrucción resulta más sencilla para el RAID de nivel 1, ya que se pueden copiar los datos de otro disco; para los otros niveles hay que acceder a todos los demás discos de la disposición para reconstruir los datos del disco averiado. El **rendimiento de la reconstrucción** de un sistema RAID puede ser un factor importante si se necesita que los datos estén disponibles ininterrumpidamente, como ocurre en los sistemas de bases de datos de alto rendimiento. Además, dado que el tiempo de reconstrucción puede suponer una parte importante del tiempo de reparación, el rendimiento de la reconstrucción también influye en el tiempo medio entre pérdidas de datos.

El nivel 0 de RAID se utiliza en aplicaciones de alto rendimiento en las que la seguridad de los datos no es crítica. Dado que los niveles 2 y 4 de RAID se subsumen en el 3 y en el 5, respectivamente, la elección del nivel RAID se limita a los niveles restantes. La distribución de bits (nivel 3) es inferior a la distribución de bloques (nivel 5), ya que esta permite velocidades de transferencia de datos similares para las transferencias de gran tamaño y emplea menos discos para las pequeñas. Para las transferencias de pequeño tamaño, el tiempo de acceso al disco es el factor dominante, por lo que disminuye la ventaja de las operaciones de lectura paralelas. De hecho, el nivel 3 puede presentar un peor rendimiento que el nivel 5 para transferencias de pequeño tamaño, ya que solo se completan cuando se han encontrado los sectores correspondientes de todos los discos; la latencia media de la disposición de discos se comporta, por tanto, de forma muy parecida a la máxima latencia en el caso de un solo disco, lo que anula las ventajas de la mayor velocidad de transferencia. Muchas implementaciones RAID no soportan actualmente el nivel 6, pero ofrece mayor fiabilidad que el nivel 5 y se puede utilizar en aplicaciones en las que la seguridad de los datos sea muy importante.

La elección entre el nivel 1 y el nivel 5 de RAID es más difícil de tomar. El nivel 1 se utiliza mucho en aplicaciones como el almacenamiento de archivos de registro histórico en sistemas de bases de datos, ya que ofrece el mejor rendimiento para escritura. El nivel 5 tiene menos sobrecarga de almacenamiento que el nivel 1, pero presenta una mayor sobrecarga temporal en las operaciones de escritura. Para las aplicaciones en las que los datos se leen con frecuencia y se escriben raramente, el nivel 5 es la opción preferida.

La capacidad de almacenamiento en disco ha aumentado anualmente más del cincuenta por ciento durante muchos años, y el coste por byte ha disminuido a la misma velocidad. En consecuencia, para muchas aplicaciones existentes de bases de datos con requisitos de almacenamiento moderados, el coste económico del almacenamiento adicional en disco necesario para la creación de imágenes ha pasado a ser relativamente pequeño (sin embargo, el coste económico adicional sigue siendo un aspecto significativo para las aplicaciones de almacenamiento intensivo como el almacenamiento de datos de vídeo). La velocidad de acceso ha mejorado mucho más lentamente (del orden de un factor tres en diez años), mientras que el número de operaciones E/S necesarias por segundo se ha incrementado enormemente, especialmente para los servidores de aplicaciones web.

El nivel 5 de RAID, que incrementa el número de operaciones E/S necesarias para escribir un solo bloque lógico, sufre una penalización de tiempo significativa en términos del rendimiento de escritura. El nivel 1 de RAID es, por tanto, la elección adecuada para muchas aplicaciones con requisitos moderados de almacenamiento y grandes de E/S.

Los diseñadores de sistemas RAID también tienen que tomar otras decisiones. Por ejemplo, la cantidad de discos que habrá en la disposición y los bits que debe proteger cada bit de paridad. Cuantos más discos haya en la disposición, mayores serán las velocidades de transferencia de datos, pero el sistema será más caro. Cuantos más bits proteja cada bit de paridad, menor será la necesidad adicional de espacio debida a los bits de paridad, pero habrá más posibilidades de que falle un segundo disco antes de que el primer disco averiado esté reparado y de que eso dé lugar a pérdidas de datos.

10.3.5. Aspectos del hardware

Otro aspecto que debe tenerse en cuenta en la elección de implementaciones RAID es el hardware. RAID se puede implementar sin cambios en el nivel hardware, modificando solo el software. Estas implementaciones se conocen como **RAID software.** Sin embargo, se pueden obtener ventajas significativas al crear un hardware específico para dar soporte a RAID, que se describen a continuación; los sistemas con soporte hardware especial se denominan sistemas **RAID hardware.**

Las implementaciones RAID hardware pueden utilizar RAM no volátil para registrar las operaciones de escritura antes de llevarlas a cabo. En caso de fallo de corriente, cuando el sistema se restaura recupera la información relativa a las operaciones de escritura incompletas de la memoria RAM no volátil y completa esas operaciones. Sin ese soporte hardware, hay que llevar a cabo un trabajo adicional para detectar los bloques que se han escrito parcialmente antes del fallo de corriente (véase el Ejercicio práctico 10.3).

Aunque todas las escrituras se hayan realizado correctamente, hay una pequeña probabilidad de que un sector de un disco no se pueda leer en algún momento. Las razones para la pérdida de datos en sectores individuales pueden ir desde defectos de fabricación a la corrupción de datos en una pista cuando se escribe una adyacente de forma repetida. Este tipo de pérdida de datos que se escribieron correctamente antes a veces se llama *fallo latente* o *bit menguante*. Cuando se producen estos fallos se detectan pronto y los datos se pueden recuperar con el resto de discos de la disposición RAID. Sin embargo, si el fallo se mantiene sin detectar, un fallo en un único disco puede conllevar la pérdida de datos si un sector de uno de los discos tiene fallos latentes.

Para minimizar la posibilidad de pérdida de los datos, los buenos controladores RAID realizan un **rastreo,** es decir, en los periodos en los que los discos están ociosos, se leen sectores del disco y si se descubre uno no legible, se recuperan los datos con el resto de discos de la disposición RAID, y el sector se vuelve a escribir (si el sector tiene un daño físico, el controlador de disco puede reasignar la dirección del sector lógico a otra ubicación física diferente en el disco).

Algunas implementaciones RAID permiten el **intercambio en caliente;** esto es, se pueden retirar los discos averiados y sustituirlos por otros nuevos sin desconectar la corriente. El intercambio en caliente reduce el tiempo medio de reparación, ya que los cambios de disco no necesitan esperar a que se pueda apagar el sistema. De hecho, hoy en día muchos sistemas críticos trabajan con un horario 24×7; esto es, funcionan 24 horas al día, 7 días a la semana, lo que no ofrece ningún momento para apagar el sistema y cambiar los discos averiados. Además, muchas implementaciones RAID asignan un disco de repuesto para cada disposición (o para un conjunto de disposiciones de disco). Si falla algún disco, el disco de repuesto se utiliza como sustituto de manera inmediata. En consecuencia, el tiempo medio de reparación se reduce notablemente, lo que minimiza la posibilidad de pérdida de datos. El disco averiado se puede reemplazar cuando resulte más conveniente.

La fuente de alimentación, el controlador de disco o, incluso, la interconexión del sistema en un sistema RAID pueden ser el punto de fallo que detiene el funcionamiento del sistema RAID. Para evitar esta posibilidad, las buenas implementaciones RAID tienen varias fuentes de alimentación redundantes (con baterías de respaldo que les permiten seguir funcionando aunque se corte la corriente). Estos sistemas RAID tienen varias interfaces de disco y varias interconexiones para conectar el sistema RAID con el sistema informático (o con la red de sistemas informáticos). En consecuencia, el fallo de un solo componente no detiene el funcionamiento del sistema RAID.

10.3.6. Otras aplicaciones de RAID

Los conceptos de RAID se han generalizado a otros dispositivos de almacenamiento, como los conjuntos de cintas, e incluso a la transmisión de datos mediante sistemas inalámbricos. Cuando se aplican a los conjuntos de cintas, las estructuras RAID pueden recuperar datos aunque se haya dañado una de las cintas de la disposición. Cuando se aplican a la transmisión de datos, cada bloque de datos se divide en unidades menores y se transmite junto con una unidad de paridad; si, por algún motivo, no se recibe alguna de las unidades, se puede reconstruir a partir del resto.

10.4. Almacenamiento terciario

Puede que en los sistemas de bases de datos de gran tamaño parte de los datos tengan que residir en un almacenamiento terciario. Los dos medios de almacenamiento terciario más habituales son los discos ópticos y las cintas magnéticas.

10.4.1. Discos ópticos

Los discos compactos han sido un medio popular de distribución de software, datos multimedia como el sonido y las imágenes, así como otra información editada de manera electrónica. Tienen una capacidad de almacenamiento de 640 a 700 megabytes y resultan baratos de producir en masa. Actualmente, los discos de vídeo digital (DVD) han reemplazado a los discos compactos en las aplicaciones que necesitan grandes cantidades de datos. Los discos de formato DVD-5 pueden almacenar 4,7 gigabytes de datos (en una capa de grabación), mientras que los discos de formato DVD-9 pueden almacenar 8,5 gigabytes de datos (en dos capas de grabación). La grabación en las dos caras del disco ofrece capacidades incluso mayores; los formatos DVD-10 y DVD-18, que son las versiones de doble cara de DVD-5 y DVD-9, pueden almacenar 9,4 y 17 gigabytes, respectivamente. El formato *Blu-ray DVD* tiene una capacidad significativamente mayor de 27 a 54 gigabytes por disco.

Las unidades de CD y de DVD presentan tiempos de búsqueda mucho mayores (100 milisegundos es un valor habitual) que las unidades de disco magnético, debido a que el dispositivo de cabezas es más pesado. Las velocidades rotacionales suelen ser menores, aunque las unidades de CD y de DVD más rápidas alcanzan alrededor de 3.000 revoluciones por minuto, que son comparables a las velocidades de los discos magnéticos de gama baja. Las velocidades rotacionales de las unidades de CD se correspondían inicialmente con las normas de los CD de sonido; y las velocidades de las unidades de DVD con las de los DVD de vídeo, pero las unidades actuales rotan a varias veces la velocidad indicada por dichas normas.

Las velocidades de transferencia de datos son algo menores que para los discos magnéticos. Las unidades de CD actuales leen entre 3 y 6 megabytes por segundo; y las de DVD, de 8 a 20 megabytes por segundo. Al igual que las unidades de discos magnéticos, los discos ópticos almacenan más datos en las pistas exteriores y menos en las interiores. La velocidad de transferencia de las unidades ópticas se caracteriza por $n\times$, que significa que la unidad soporta transferencias a n veces la velocidad indicada por la norma; las velocidades de $50\times$ para CD y de $16\times$ para DVD son frecuentes hoy en día.

Las versiones de los discos ópticos en las que solo se puede escribir una vez (CD-R, DVD-R y DVD+R) son populares para la distribución de datos y, en especial, para el almacenamiento de datos con fines de archivo, debido a que tienen una gran capacidad, una vida más larga que los discos magnéticos y pueden retirarse de la unidad y almacenarse en un lugar remoto. Dado que no pueden sobrescribirse, se pueden utilizar para almacenar información que no se deba modificar, como los registros de auditoría. Las versiones en las que se puede escribir más de una vez (CD-RW, DVD-RW, DVD+RW y DVD-RAM) también se emplean para archivo.

Los **cambiadores de discos** (*jukebox*) son dispositivos que guardan gran número de discos ópticos (hasta varios cientos) y los cargan automáticamente a petición de los usuarios en unas cuantas unidades (usualmente entre una y diez). La capacidad de almacenamiento agregada de estos sistemas puede ser de muchos terabytes. Cuando se accede a un disco, un brazo mecánico lo carga en la unidad desde una estantería (antes hay que volver a colocar en la estantería el disco que se hallaba en la unidad). El tiempo de carga y descarga suele ser del orden de unos pocos segundos (mucho mayor que los tiempos de acceso a disco).

10.4.2. Cintas magnéticas

Aunque las cintas magnéticas son relativamente permanentes y pueden albergar grandes volúmenes de datos, resultan lentas en comparación con los discos magnéticos y ópticos, y lo que es aún más importante, la cinta magnética está limitada al acceso secuencial. Por tanto, no puede proporcionar acceso aleatorio para los requisitos de almacenamiento secundario, aunque históricamente, antes del uso de los discos magnéticos, las cintas se utilizaban como medio de almacenamiento secundario.

Las cintas se emplean principalmente para realizar copias de seguridad, para el almacenamiento de la información poco utilizada y como medio sin conexión para la transferencia de información de un sistema a otro. Las cintas también se utilizan para almacenar grandes volúmenes de datos, como los datos de vídeo o de imágenes, a los que no es necesario acceder rápidamente o que son tan voluminosos que su almacenamiento en disco magnético resultaría demasiado caro.

Cada cinta se guarda en una bobina y se enrolla o desenrolla por delante de una cabeza de lectura y escritura. El desplazamiento hasta el punto correcto de la cinta puede tardar segundos o, incluso, minutos, en vez de milisegundos; una vez en posición, sin embargo, las unidades de cinta pueden escribir los datos con densidades y velocidades que se aproximan a las de las unidades de disco. La capacidad varía en función de la longitud y de la anchura de la cinta, así como de la densidad con la que la cabeza pueda leer y escribir. El mercado ofrece actualmente una amplia variedad de formatos de cinta. La capacidad disponible hoy en día varía de unos pocos gigabytes con el formato DAT (*digital audio tape*: cinta de audio digital), de 10 a 40 gigabytes con el formato DLT (*digital linear tape*: cinta lineal digital), de 100 gigabytes en adelante con el formato **Ultrium,** hasta 330 gigabytes con los formatos de cinta de **exploración helicoidal de Ampex.** Las velocidades de transferencia de datos van de unos pocos a decenas de megabytes por segundo.

Las unidades de cinta son bastante fiables, y los buenos sistemas de unidades de cinta realizan una lectura de los datos recién escritos para asegurarse de que se han grabado correctamente. Sin embargo, las cintas presentan límites en cuanto al número de veces que se pueden leer o escribir con fiabilidad.

Los **cambiadores de cintas,** al igual que los cambiadores de discos ópticos, tienen gran número de cintas y unas cuantas unidades en las que se pueden montar esas cintas; se utilizan para guardar grandes volúmenes de datos, que pueden llegar a varios petabytes (10^{15} bytes), con tiempos de acceso que varían entre unos segundos y unos pocos minutos. Entre las aplicaciones que necesitan estas ingentes cantidades de datos están los sistemas de imágenes que reúnen los datos de los satélites de teledetección y las grandes videotecas de las emisoras de televisión.

Algunos formatos de cinta (como el formato Accelis) soportan tiempos de búsqueda más rápidos (del orden de decenas de segundos) y están pensados para las aplicaciones que recuperan información mediante cambiadores de cintas. La mayor parte del resto de los formatos de cinta proporcionan mayores capacidades, a cambio de un acceso más lento; estos formatos resultan idóneos para las copias de seguridad de los datos, en las que no son importantes unos tiempos de búsqueda rápidos.

Las unidades de cinta no han podido competir con la enorme mejora de capacidad de las unidades de disco y la correspondiente reducción del coste de almacenamiento. Aunque el coste de las cintas es bajo, el de las unidades y las bibliotecas de cintas es significativamente superior que el coste de las unidades de disco: una biblioteca de cintas capaz de almacenar algunos terabytes puede costar decenas de miles de euros Las copias de seguridad en unidades de disco se han convertido en una alternativa rentable a las copias en cinta para gran número de aplicaciones.

10.5. Organización de los archivos

Una base de datos se hace corresponder en un número de archivos diferentes que mantiene el sistema operativo. Estos archivos residen permanentemente en discos. Los **archivos** se organizan lógicamente como secuencias de registros. Esos registros se corresponden con los bloques del disco. Los archivos constituyen un elemento fundamental de los sistemas operativos, por lo que se supone la existencia de un *sistema de archivos* subyacente. Hay que tomar en consideración diversas maneras de representar los modelos lógicos de datos en términos de los archivos.

Los archivos se dividen lógicamente en unidades de almacenamiento de tamaño fijo llamados **bloques,** que son las unidades tanto de asignación de almacenamiento como de transferencia de datos. La mayoría de las bases de datos usan tamaños de bloque de 4 a 8 kilobytes, pero muchas bases de datos permiten definir el tamaño del bloque cuando se crea un ejemplar de la base de datos. En algunas aplicaciones pueden ser útiles tamaños de bloque mayores.

Un bloque puede contener varios registros; el número exacto de registros que contiene depende de la organización física de los datos que se use. Se supone que *ningún registro es mayor que un bloque*. Esta suposición es realista para la mayoría de las aplicaciones de procesamiento de datos, como nuestro ejemplo de la universidad. Existen algunos tipos de datos que son grandes, como las imágenes, que pueden ser significativamente más grandes que un bloque. Más adelante se tratará cómo manejar estos grandes elementos de datos, en la Sección 10.5.2, almacenando estos grandes elementos de datos de forma separada y guardando una referencia al dato en el registro.

Además, se requiere que *cada registro se encuentre completo en un único bloque*; es decir, ningún registro se encuentra parcialmente en un bloque y parcialmente en otro. Esta restricción simplifica y acelera el acceso a los elementos de datos.

En una base de datos relacional, las tuplas de relaciones diferentes suelen ser de tamaños diversos. Un enfoque de la correspondencia entre base de datos y archivos es utilizar varios archivos y guardar los registros de la misma longitud en un mismo archivo. Una alternativa es estructurar los archivos de modo que puedan aceptar registros de longitudes diferentes; no obstante, los archivos con registros de longitud fija son más sencillos de implementar que los que tienen registros de longitud variable. Muchas de las técnicas empleadas para los primeros pueden aplicarse a los de longitud variable. Por tanto, se comienza considerando archivos con registros de longitud fija.

10.5.1. Registros de longitud fija

A modo de ejemplo, considere un archivo con registros *profesor* de la base de datos de la universidad. Cada registro de este archivo se define (en pseudocódigo) de la siguiente manera:

```
type profesor = record
        ID varchar (5);
        nombre varchar(20);
        nombre_dept varchar (20);
        sueldo numeric (8,2);
    end
```

Suponga que cada carácter ocupa un byte y que los valores de tipo numeric(8,2) ocupan ocho bytes. Suponga que en lugar de asignar un número variable de bytes para los atributos *ID*, *nombre* y *nombre_dept* se asigna el número máximo de bytes que puede contener cada atributo. Entonces, el registro *profesor* tiene 53 bytes de longitud. Un enfoque sencillo es utilizar los primeros 53 bytes para el primer registro, los 53 bytes siguientes para el segundo, y así sucesivamente (Figura 10.4).

Sin embargo, hay dos problemas con este sencillo enfoque:

1. A menos que el tamaño de los bloques sea múltiplo de 53 (lo que resulta improbable), algún registro se saltará los límites de los bloques. Es decir, parte del registro se guardará en un bloque y parte en otro. Harán falta, por tanto, dos accesos a bloques para leer o escribir esos registros.
2. Resulta difícil borrar registros de esta estructura. Hay que rellenar el espacio ocupado por el registro que se va a borrar con algún otro registro del archivo, o tener alguna manera de marcar los registros borrados para poder pasarlos por alto.

registro 0	10101	Srinivasan	Informática	65000
registro 1	12121	Wu	Finanzas	90000
registro 2	15151	Mozart	Música	40000
registro 3	22222	Einstein	Física	95000
registro 4	32343	El Said	Historia	60000
registro 5	33456	Gold	Física	87000
registro 6	45565	Katz	Informática	75000
registro 7	58583	Califieri	Historia	62000
registro 8	76543	Singh	Finanzas	80000
registro 9	76766	Crick	Biología	72000
registro 10	83821	Brandt	Informática	92000
registro 11	98345	Kim	Electrónica	80000

Figura 10.4. Archivo con registro *profesor*.

Para evitar el primer problema, solo se asignan tantos registros en un bloque como quepan enteros en el bloque (este número se puede calcular fácilmente dividiendo el tamaño del bloque entre el tamaño del registro y descartando la parte decimal). Cualquier resto de bytes de cada bloque se deja sin usar.

Cuando se borra un registro, se puede desplazar el situado a continuación al espacio que ocupaba el registro borrado y hacer lo mismo con los demás, hasta que todos los registros situados a continuación del borrado se hayan desplazado hacia delante (Figura 10.5). Este tipo de enfoque necesita desplazar gran número de registros. Puede que fuera más sencillo desplazar simplemente el último registro del archivo al espacio ocupado por el registro borrado (Figura 10.6).

No resulta deseable desplazar los registros para que ocupen el espacio liberado por los registros borrados, ya que para hacerlo se necesitan accesos adicionales a los bloques. Dado que las operaciones de inserción tienden a ser más frecuentes que las de borrado, resulta aceptable dejar libre el espacio del registro borrado, y esperar a una inserción posterior antes de volver a utilizar ese espacio. No basta con una simple marca en el registro borrado, ya que resulta difícil hallar el espacio disponible mientras se realiza una inserción. Por tanto, hay que introducir una estructura adicional.

Al comienzo del archivo se asigna cierto número de bytes como **cabecera del archivo.** La cabecera contiene gran variedad de información sobre el archivo. Por ahora, todo lo que hace falta guardar es la dirección del primer registro cuyo contenido se haya borrado. Se utiliza este primer registro para guardar la dirección del segundo registro disponible, y así sucesivamente. De manera intuitiva se pueden considerar estas direcciones guardadas como *punteros*, dado que indican la posición de un registro. Los registros borrados, por tanto, forman una lista enlazada a la que se suele denominar **lista libre.** La Figura 10.7 muestra el archivo de la Figura 10.4, con la lista libre, después de haberse borrado los registros 1, 4 y 6.

Al insertar un registro nuevo se utiliza el registro al que apunta la cabecera. Se modifica el puntero de la cabecera para que señale al siguiente registro disponible. Si no hay espacio disponible, se añade el nuevo registro al final del archivo.

La inserción y el borrado de archivos con registros de longitud fija es sencillo de implementar, dado que el espacio que deja libre cada registro borrado es exactamente el mismo que se necesita para insertar otro registro. Si se permiten en un archivo registros de longitud variable, esta coincidencia no se mantiene. Puede que el registro insertado no quepa en el espacio liberado por el registro borrado, o que solo llene una parte.

registro 0	10101	Srinivasan	Informática	65000
registro 1	12121	Wu	Finanzas	90000
registro 2	15151	Mozart	Música	40000
registro 4	32343	El Said	Historia	60000
registro 5	33456	Gold	Física	87000
registro 6	45565	Katz	Informática	75000
registro 7	58583	Califieri	Historia	62000
registro 8	76543	Singh	Finanzas	80000
registro 9	76766	Crick	Biología	72000
registro 10	83821	Brandt	Informática	92000
registro 11	98345	Kim	Electrónica	80000

Figura 10.5. Archivo de la Figura 10.4, en el que se ha borrado el registro 3 y se han desplazado todos los registros.

registro 0	10101	Srinivasan	Informática	65000
registro 1	12121	Wu	Finanzas	90000
registro 2	15151	Mozart	Música	40000
registro 11	98345	Kim	Electrónica	80000
registro 4	32343	El Said	Historia	60000
registro 5	33456	Gold	Física	87000
registro 6	45565	Katz	Informática	75000
registro 7	58583	Califieri	Historia	62000
registro 8	76543	Singh	Finanzas	80000
registro 9	76766	Crick	Biología	72000
registro 10	83821	Brandt	Informática	92000

Figura 10.6. Archivo de la Figura 10.4, en el que se ha borrado el registro 3 y se ha movido el registro final.

10.5.2. Registros de longitud variable

Los registros de longitud variable surgen de varias maneras en los sistemas de bases de datos:

- Almacenamiento de varios tipos de registros en un mismo archivo.
- Tipos de registros que permiten longitudes variables para uno o varios de los campos.
- Tipos de registros que permiten campos repetidos, como los arrays o los multiconjuntos.

cabecera				
registro 0	10101	Srinivasan	Informática	65000
registro 1				
registro 2	15151	Mozart	Música	40000
registro 3	22222	Einstein	Física	95000
registro 4				
registro 5	33456	Gold	Física	87000
registro 6				
registro 7	58583	Califieri	Historia	62000
registro 8	76543	Singh	Finanzas	80000
registro 9	76766	Crick	Biología	72000
registro 10	83821	Brandt	Informática	92000
registro 11	98345	Kim	Electrónica	80000

Figura 10.7. Archivo de la Figura 10.4, con la lista libre tras el borrado de los registros 1, 4 y 6.

Existen diferentes técnicas para implementar los registros de longitud variable. En cualquiera de ellas hay que resolver dos problemas:

- Cómo representar un registro único de forma que se puedan extraer fácilmente los atributos individuales.
- Cómo guardar registros de tamaño variable en un bloque, de forma que se puedan extraer fácilmente.

La representación de los registros con atributos de longitud variable normalmente consta de dos partes: una parte inicial con los atributos de longitud fija, seguida de los datos para los atributos de longitud variable. A los atributos de longitud fija, como los valores numéricos, las fechas o las cadenas de caracteres de tamaño fijo se les asigna tantos bytes como requieran para guardar sus valores. Los atributos de longitud variable, como los de tipo varchar, se representan en la parte inicial del registro como un par (*desplazamiento*, *longitud*), en el que *desplazamiento* indica dónde comienzan los datos de dicho atributo en el registro y *longitud* es la longitud en bytes del atributo de longitud variable. Los valores de estos atributos se almacenan consecutivamente, tras la parte de longitud fija del registro. Por tanto, la parte inicial del registro almacena una información de longitud fija sobre cada atributo, ya sea de longitud fija o variable.

Como ejemplo de esta representación de registro se puede ver la Figura 10.8. Esta figura muestra un registro *profesor*, cuyos primeros tres atributos *ID*, *nombre* y *nombre_dept* son cadenas de longitud variable y cuyo cuarto atributo, *sueldo*, es un numero de longitud fija. Suponemos que los valores de desplazamiento y de longitud se guardan en dos bytes cada uno, lo que hace un total de 4 bytes para cada atributo. Se supone que el atributo *sueldo* se almacena en 8 bytes, y cada cadena de caracteres tiene tantos bytes como caracteres.

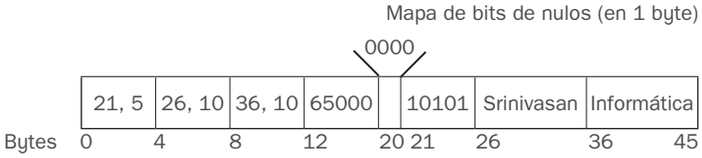


Figura 10.8. Representación de un registro de longitud variable.

En la figura también se muestra el uso de un **mapa de bits de nulos,** que indica los atributos del registro que tienen el valor nulo. En este registro concreto, si el sueldo fuese nulo, el cuarto bit del mapa de bits de nulos se pondría a 1, y el valor de *sueldo* almacenado en los bytes 12 al 19 se ignoraría. Como el registro tiene cuatro atributos, el mapa de bits de nulos para este registro cabe en un byte, aunque pueden necesitarse más bytes si hay más atributos. En algunas representaciones, el mapa de bits de nulos se almacena al principio del registro, y para los atributos que son nulos no se guarda información (valor, ni desplazamiento/longitud). Esta representación ahorra algún espacio de almacenamiento, con el coste de trabajo adicional para extraer los atributos del registro. Esta representación en concreto es útil para algunas aplicaciones en las que los registros tienen un gran número de campos, la mayor parte de ellos nulos.

A continuación se trata el problema de almacenar registros de longitud variable en un bloque. La **estructura de páginas con ranuras** se utiliza habitualmente para organizar los registros en bloques, y puede verse en la Figura 10.9.[3] Hay una cabecera al principio de cada bloque, que contiene la siguiente información:

1. El número de elementos del registro de la cabecera.
2. El final del espacio vacío del bloque.
3. Un array cuyas entradas contienen la ubicación y el tamaño de cada registro.

3 Aquí, «página» es sinónimo de «bloque».

Los registros reales se ubican en el bloque *de manera contigua*, empezando por el final. El espacio libre dentro del bloque es contiguo, entre la última entrada del array de la cabecera y el primer registro. Si se inserta un registro, se le asigna espacio al final del espacio libre y se añade a la cabecera una entrada que contiene su tamaño y su ubicación.

Si se borra un registro, se libera el espacio que ocupa y se da el valor de *borrado* a su entrada (por ejemplo, se le da a su tamaño el valor de –1). Además, se desplazan los registros del bloque situados antes del registro borrado, de modo que se ocupe el espacio libre generado por el borrado y todo el espacio libre vuelve a hallarse entre la última entrada del array de la cabecera y el primer registro. También se actualiza de manera adecuada el puntero de final del espacio libre de la cabecera. Se puede aumentar o disminuir el tamaño de los registros mediante técnicas parecidas, siempre y cuando quede espacio en el bloque. El coste de trasladar los registros no es demasiado elevado, ya que el tamaño del bloque es limitado: un valor habitual de 4 a 8 kilobytes.

La estructura de páginas con ranuras necesita que no haya punteros que señalen directamente a los registros. Por el contrario, los punteros deben apuntar a la entrada de la cabecera que contiene la ubicación verdadera del registro. Este nivel de indirección permite que se desplacen los registros para evitar la fragmentación del espacio del bloque al tiempo que permite los punteros indirectos al registro.

Las bases de datos a menudo almacenan datos que pueden ser mucho más grandes que los bloques del disco. Por ejemplo, las imágenes o las grabaciones de sonido pueden tener un tamaño de varios megabytes, mientras que los objetos de vídeo pueden llegar a los gigabytes. Recuerde que SQL soporta los tipos **blob** y **clob,** que almacenan objetos de gran tamaño de los tipos binario y carácter, respectivamente.

La mayor parte de las bases de datos relacionales limitan el tamaño de los registros para que no superen el tamaño de los bloques, con objeto de simplificar la gestión de la memoria intermedia y del espacio libre. Los objetos de gran tamaño y los campos largos suelen guardarse en archivos especiales (o conjuntos de archivos) en lugar de almacenarse con los otros atributos de pequeño tamaño de los registros en los que aparecen. Entonces se guarda un puntero (lógico) al objeto en el registro que contiene el objeto de gran tamaño. Los objetos de gran tamaño se suelen representar en organizaciones de archivos de árboles B^+, que se estudian en la Sección 11.4.1, que permiten leer el objeto completo o rangos concretos de bytes del mismo, así como insertar y borrar partes del objeto.

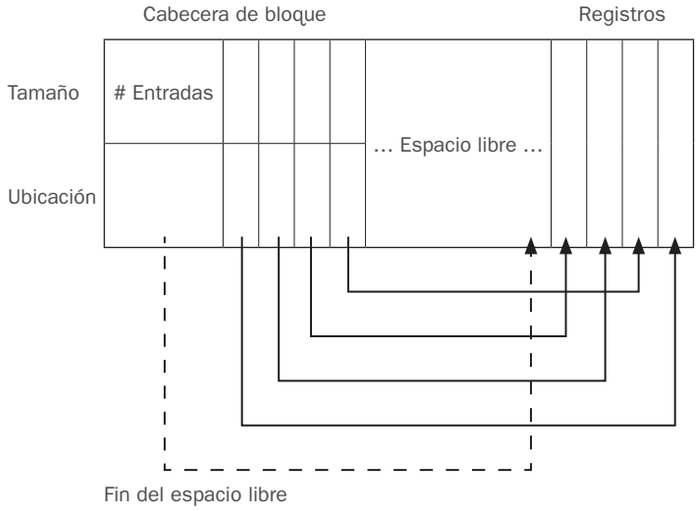


Figura 10.9. Estructura de páginas con ranuras.

10.6. Organización de los registros en archivos

Hasta ahora se ha estudiado la manera en que se representan los registros en la estructura de archivo. Las relaciones son conjuntos de registros. Dado un conjunto de registros, la pregunta siguiente es cómo organizarlos en archivos. A continuación se indican varias maneras de organizar los registros en archivos:

- **Organización de los archivos en montículos.** Cualquier registro se puede colocar en cualquier parte del archivo en que haya espacio suficiente. Los registros no se ordenan. Generalmente solo hay un archivo para cada relación.
- **Organización secuencial de los archivos.** Los registros se guardan en orden secuencial, según el valor de la «clave de búsqueda» de cada uno. La Sección 10.6.1 describe esta organización.
- **Organización asociativa (hash) de los archivos.** Se calcula una función de asociación *(hash)* para algún atributo de cada registro. El resultado de la función de asociación especifica el bloque del archivo en que se debe colocar cada registro. El Capítulo 11 describe esta organización, estrechamente relacionada con las estructuras para la creación de índices que se describen en ese capítulo.

10101	Srinivasan	Informática	65000	
12121	Wu	Finanzas	90000	
15151	Mozart	Música	40000	
22222	Einstein	Física	95000	
32343	El Said	Historia	60000	
33456	Gold	Física	87000	
45565	Katz	Informática	75000	
58583	Califieri	Historia	62000	
76543	Singh	Finanzas	80000	
76766	Crick	Biología	72000	
83821	Brandt	Informática	92000	
98345	Kim	Electrónica	80000	

Figura 10.10. Archivo secuencial para los registros de *profesor*.

Generalmente se emplea un archivo separado para almacenar los registros de cada relación. Sin embargo, en cada **organización de archivos en agrupaciones de varias tablas** se pueden guardar en el mismo archivo registros de relaciones diferentes; además, los registros relacionados de las diferentes relaciones se guardan en el mismo bloque, por lo que cada operación de E/S afecta a registros relacionados de todas esas relaciones. Por ejemplo, los registros de dos relaciones se pueden considerar relacionados si casan en una reunión de las dos relaciones. En la Sección 10.6.2 se describe esta organización.

10.6.1. Organización de archivos secuenciales

Los **archivos secuenciales** están diseñados para el procesamiento eficiente de los registros de acuerdo con un orden por alguna clave de búsqueda. La **clave de búsqueda** es cualquier atributo o conjunto de atributos; no tiene por qué ser la clave primaria, ni siquiera una superclave. Para permitir la recuperación rápida de los registros según el orden de la clave de búsqueda, estos se vinculan mediante punteros. El puntero de cada registro señala al siguiente registro según el orden indicado por la clave de búsqueda. Además, para minimizar el número de accesos a los bloques en el procesamiento de los archivos secuenciales, los registros se guardan físicamente en el orden indicado por la clave de búsqueda, o lo más cercano posible.

La Figura 10.10 muestra un archivo secuencial de registros de *profesor* tomado del ejemplo de nuestra universidad. En ese ejemplo, los registros se guardan de acuerdo con el orden de la clave de búsqueda; en este caso, *ID*.

La organización secuencial de archivos permite que los registros se lean de forma ordenada, lo que puede resultar útil para la visualización, así como para ciertos algoritmos de procesamiento de consultas que se estudian en el Capítulo 12.

Sin embargo, resulta difícil mantener el orden físico secuencial a medida que se insertan y se borran registros, dado que resulta costoso desplazar muchos registros como consecuencia de una sola operación de inserción o de borrado. Se puede gestionar el borrado mediante cadenas de punteros, como ya se ha visto. Para la inserción se aplican las reglas siguientes:

1. Localizar el registro del archivo que precede al registro que se va a insertar según el orden de la clave de búsqueda.
2. Si existe algún registro vacío (es decir, un espacio que haya quedado libre después de una operación de borrado) dentro del mismo bloque de ese registro, el registro nuevo se insertará ahí. En caso contrario, el nuevo registro se insertará en un *bloque de desbordamiento*. En cualquier caso, hay que ajustar los punteros para vincular los registros según el orden de la clave de búsqueda.

La Figura 10.11 muestra el archivo de la Figura 10.10 después de la inserción del registro (32222, Verdi, Música, 48000). La estructura de la Figura 10.11 permite la inserción rápida de registros nuevos, pero obliga a las aplicaciones de procesamiento de archivos secuenciales a procesar los registros en un orden que no coincide con el físico.

Si hay que guardar un número relativamente pequeño de registros en los bloques de desbordamiento, este enfoque funciona bien. No obstante, con el tiempo, la correspondencia entre el orden de la clave de búsqueda y el físico puede perderse totalmente, en cuyo caso el procesamiento secuencial acaba siendo mucho menos eficiente. En este momento, se debe **reorganizar** el archivo de modo que vuelva a estar físicamente en orden secuencial. Estas reorganizaciones resultan costosas y deben realizarse en momentos en los que la carga del sistema sea baja. La frecuencia con la que las reorganizaciones son necesarias depende de la frecuencia de inserción de registros nuevos. En el caso extremo en que rara vez se produzcan inserciones, es posible mantener siempre el archivo en el orden físico correcto. En ese caso, el campo puntero de la Figura 10.10 no es necesario.

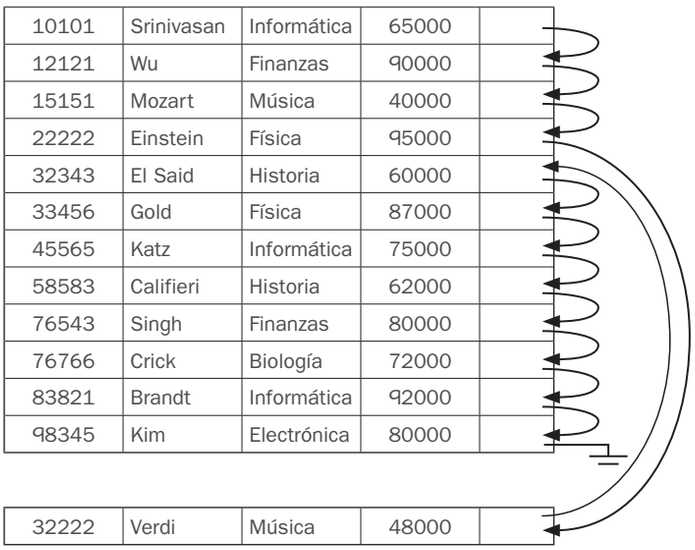


Figura 10.11. Archivo secuencial después de una inserción.

10.6.2. Organización de archivos en agrupaciones de varias tablas

Muchos sistemas de bases de datos relacionales guardan cada relación en un archivo diferente, de modo que puedan aprovechar completamente el sistema de archivos proporcionado por el sistema operativo. Generalmente, las tuplas de cada relación se pueden representar como registros de longitud fija. Por tanto, se puede hacer que las relaciones se correspondan con una estructura de archivos sencilla. Esta implementación sencilla de los sistemas de bases de datos relacionales resulta adecuada para las implementaciones de bajo coste de las bases de datos como, por ejemplo, los sistemas empotrados o los dispositivos portátiles. En estos sistemas, el tamaño de la base de datos es pequeño, por lo que se obtiene poco provecho de una estructura de archivos avanzada. Además, en esos entornos es fundamental que el tamaño total del programa del sistema de bases de datos sea pequeño. La estructura de archivos sencilla reduce la cantidad de código necesaria para implementar el sistema.

Este enfoque sencillo de la implementación de las bases de datos relacionales resulta menos satisfactorio a medida que aumenta el tamaño de la base de datos. Ya se ha visto que se pueden obtener mejoras en el rendimiento con una cuidadosa asignación de los registros a los bloques y de los propios bloques. Por tanto, resulta evidente que una estructura de archivos más compleja puede resultar ventajosa, aunque se mantenga la estrategia de guardar cada relación en un archivo diferente.

Sin embargo, muchos sistemas de bases de datos de gran tamaño no utilizan directamente el sistema operativo subyacente para la gestión de los archivos. Por el contrario, se asigna al sistema de bases de datos un archivo de gran tamaño del sistema operativo. En este archivo, el sistema de bases de datos, que también lo administra, guarda todas las relaciones.

Aunque se guarden distintas relaciones en un único archivo, habitualmente la mayoría de las bases de datos almacenan los registros de una sola relación por bloque. De esta forma se simplifica la gestión de los datos. Sin embargo, a veces puede ser útil almacenar registros de más de una relación en un mismo bloque. Para comprender la ventaja de guardar distintas relaciones en un solo bloque, considere la siguiente consulta SQL de la base de datos de la universidad:

select *nombre_dept*, *edificio*, *presupuesto*, *ID*, *nombre*, *sueldo*
from *departamento* **natural join** *profesor*;

Esta consulta calcula una reunión de las relaciones *departamento* y *profesor*. Por tanto, por cada tupla *departamento* el sistema debe encontrar las tuplas de *profesor* con el mismo valor de *nombre_dept*. Lo ideal sería poder encontrar estos registros con la ayuda de *índices*, que se estudiarán en el Capítulo 11. Independientemente de la manera en que se encuentren esos registros, hay que transferirlos desde el disco a la memoria principal. En el peor de los casos, cada registro se hallará en un bloque diferente, lo que obligará a efectuar un proceso de lectura de bloque por cada registro necesario para la consulta.

nombre_dept	*edificio*	*presupuesto*
Informática	Taylor	100000
Física	Watson	70000

Figura 10.12. La relación *departamento*.

ID	*nombre*	*nombre_dept*	*sueldo*
10101	Srinivasan	Informática	65000
33456	Gold	Física	87000
45565	Katz	Informática	75000
83821	Brandt	Informática	92000

Figura 10.13. La relación *profesor*.

A modo de ejemplo, considere las relaciones *departamento* y *profesor* de las Figuras 10.12 y 10.13, respectivamente (por brevedad, solo se ha incluido un subconjunto de las tuplas de las relacionas que se han visto en otros ejemplos). En la Figura 10.14 se muestra una estructura de archivo diseñada para la ejecución eficiente de las consultas que implican la reunión natural de *departamento* y *profesor*. Las tuplas *profesor* para cada *ID* se guardan cerca de la tupla *departamento* para el *nombre_dept* correspondiente. Esta estructura mezcla las tuplas de dos relaciones, pero permite el procesamiento eficaz de la reunión. Cuando se lee una tupla de la relación *departamento*, se copia del disco a la memoria principal todo el bloque que contiene esa tupla. Dado que las tuplas correspondientes de *profesor* se guardan en el disco cerca de la tupla *departamento*, el bloque que contiene la tupla *departamento* también contiene tuplas de la relación *profesor* necesarias para procesar la consulta. Si un departamento tiene tantos profesores que los registros de *profesor* no caben en un solo bloque, los registros restantes aparecerán en bloques cercanos.

Una **organización de archivos en agrupaciones de varias tablas** es una organización de archivos, como la mostrada en la Figura 10.14, que almacena registros relacionados de dos o más relaciones en cada bloque. Este tipo de organización de archivos permite leer registros que satisfacen la condición de reunión en un solo proceso de lectura de bloques. Por tanto, esta consulta concreta se puede procesar de manera más eficiente.

En la representación de la Figura 10.14, se omite el atributo *nombre_dept* de los registros *profesor*, ya que se pueden inferir del registro *departamento* asociado; en algunas implementaciones se puede mantener el atributo, para simplificar el acceso al mismo. Se supone que cada registro contiene el identificador de la relación a la que pertenece, aunque no se muestre en la Figura 10.14.

El empleo de la agrupación de varias tablas en un único archivo ha mejorado el procesamiento de una reunión dada (de *departamento* y *profesor*), pero el procesamiento de otras consultas será más lento. Por ejemplo:

select *
from *departamento*;

necesita más accesos a los bloques que con el esquema en el que cada relación se guardaba en un archivo diferente, ya que cada bloque contiene ahora significativamente menos registros *departamento*. Para encontrar eficientemente todas las tuplas de la relación *departamento* en la estructura de la Figura 10.14 se pueden enlazar todos los registros de esa relación mediante punteros, tal y como se muestra en la Figura 10.15.

El empleo de la agrupación de varias tablas depende de los tipos de consulta que el diseñador de la base de datos considere más frecuentes. Un uso cuidadoso de la agrupación de varias tablas puede producir mejoras significativas de rendimiento en el procesamiento de las consultas.

Informática	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Física	Watson	70000
33456	Gold	87000

Figura 10.14. Estructura de archivo en agrupaciones de varias tablas.

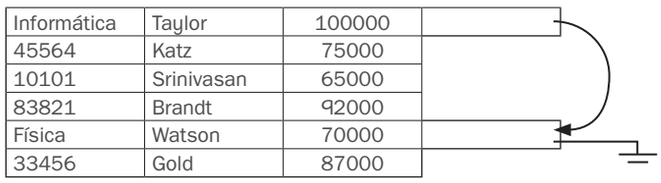


Figura 10.15. Estructura de archivo en agrupaciones de varias tablas con cadenas de punteros.

10.7. Almacenamiento con diccionarios de datos

Hasta ahora solo se ha considerado la representación de las propias relaciones. Un sistema de bases de datos relacionales necesita tener datos *sobre* las relaciones, como puede ser su esquema. En general, estos «datos sobre los datos» se denominan **metadatos.**

Los esquemas relacionales y otros metadatos sobre las relaciones se almacenan en una estructura llamada **diccionario de datos** o **catálogo del sistema.** Entre los tipos de información que debe guardar el sistema figuran los siguientes:

- El nombre de las relaciones.
- El nombre de los atributos de cada relación.
- El dominio y la longitud de los atributos.
- El nombre de las vistas definidas en la base de datos y la definición de esas vistas.
- Las restricciones de integridad (por ejemplo, las restricciones de las claves).

Además, muchos sistemas guardan los siguientes datos de los usuarios del sistema:

- El nombre de los usuarios autorizados.
- La autorización y la información sobre las cuentas de los usuarios.
- Las contraseñas u otra información utilizada para autentificar a los usuarios.

Además, la base de datos puede guardar información estadística y descriptiva sobre las relaciones, como:

- El número de tuplas de cada relación.
- El método de almacenamiento utilizado para cada relación (por ejemplo, con agrupaciones o sin agrupaciones).

El diccionario de datos puede también tener en cuenta la organización del almacenamiento (secuencial, asociativa o en montículos) de las relaciones, así como la ubicación en la que se guarda cada relación:

- Si las relaciones se almacenan en archivos del sistema operativo, el diccionario tendrá en cuenta el nombre del archivo (o archivos) que guarda cada relación.
- Si la base de datos almacena todas las relaciones en un solo archivo, puede que el diccionario tenga en cuenta los bloques que almacenan los registros de cada relación en una estructura de datos como, por ejemplo, una lista enlazada.

En el Capítulo 11, en el que se estudian los índices, se verá que hace falta guardar información sobre cada índice de cada una de las relaciones:

- El nombre del índice.
- El nombre de la relación para la que se crea.
- Los atributos sobre los que se define.
- El tipo de índice formado.

Toda esta información de metadatos constituye, en efecto, una base de datos en miniatura. Algunos sistemas de bases de datos guardan esta información utilizando código y estructuras de datos especiales. Suele resultar preferible guardar los datos sobre la base de datos como relaciones en la misma base de datos. Al utilizar las relaciones de la base de datos para guardar los metadatos del sistema, se simplifica la estructura global del sistema y se dedica toda la potencia de la base de datos a obtener un acceso rápido a los datos del sistema.

La elección exacta de la manera de representar los datos del sistema mediante las relaciones debe tomarla el diseñador del sistema. La Figura 10.16 ofrece una representación posible, con las claves primarias subrayadas. En esta representación se da por supuesto que el atributo *indice_atributos* de la relación *Metadatos_*

índices contiene una lista de uno o varios atributos, que se pueden representar mediante una cadena de caracteres como «*nombre_dept, edificio*». Por tanto, la relación *Metadatos_índices* no está en la primera forma normal; se puede normalizar, pero es probable que la representación anterior sea más eficiente en el acceso. El diccionario de datos se suele almacenar de forma no normalizada para conseguir un acceso rápido.

Cuando el sistema de bases de datos necesita recuperar los registros de una relación, debe consultar primero la relación *Metadatos_relación* para encontrar la ubicación y organización de almacenamiento de la relación y después buscar los registros usando esta información. Sin embargo, la organización del almacenamiento y la localización de la propia relación *Metadatos_relación* se deben registrar en algún lugar (por ejemplo, en el propio código de la base de datos o en una ubicación fija), ya que esa información es necesaria para encontrar el contenido de *Metadatos_relación*.

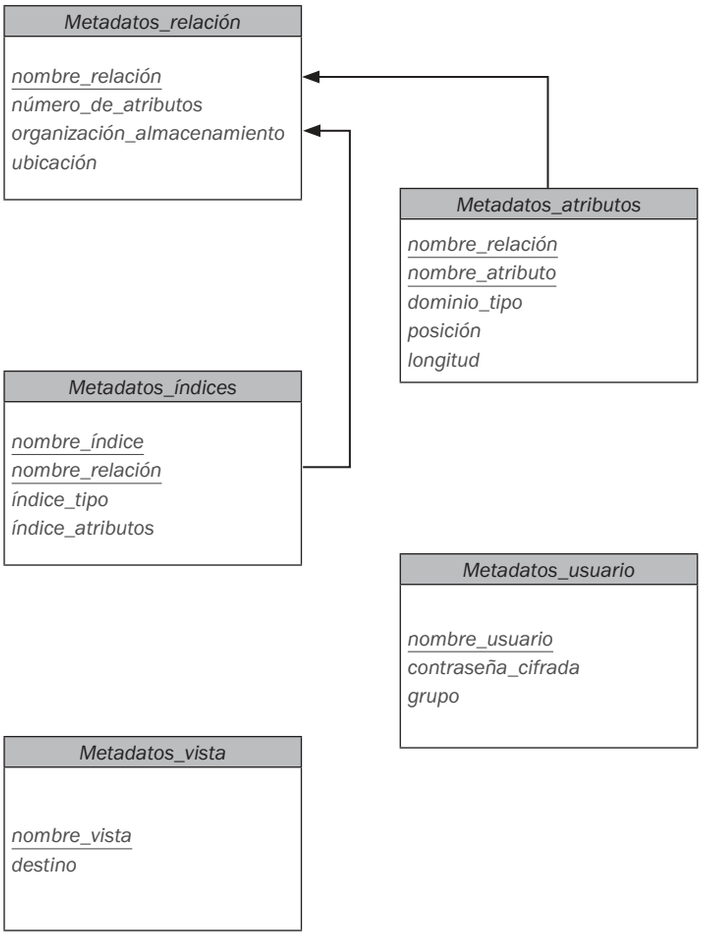


Figura 10.16. Esquema relacional representando los metadatos del sistema.

10.8. Memoria intermedia de la base de datos

Uno de los principales objetivos del sistema de bases de datos es minimizar el número de transferencias de bloques entre el disco y la memoria. Una manera de reducir el número de accesos al disco es mantener en la memoria principal tantos bloques como sea posible. El objetivo es maximizar la posibilidad de que, cuando se acceda a un bloque, ya se encuentre en la memoria principal y, por tanto, no se necesite acceder al disco.

Dado que no resulta posible mantener en la memoria principal todos los bloques, hay que gestionar la asignación del espacio allí disponible para su almacenamiento. La **memoria intermedia** (buffer) es la parte de la memoria principal disponible para el almacenamiento de las copias de los bloques del disco. Siempre se guarda en el disco una copia de cada bloque, pero esta copia puede ser una versión del bloque más antigua que la de la memoria intermedia. El subsistema responsable de la asignación del espacio de la memoria intermedia se denomina **gestor de la memoria intermedia.**

10.8.1. Gestor de la memoria intermedia

Los programas de los sistemas de bases de datos formulan solicitudes (es decir, llamadas) al gestor de la memoria intermedia cuando necesitan bloques del disco. Si el bloque ya se encuentra en la memoria intermedia, el gestor pasa al solicitante la dirección del bloque en la memoria principal. Si el bloque no se halla en la memoria intermedia, asigna en primer lugar espacio al bloque en la memoria intermedia, descartando algún otro, si hace falta, para hacer sitio al nuevo. El bloque descartado solo se vuelve a escribir en el disco si se ha modificado desde la última vez que se escribió. A continuación, el gestor de la memoria intermedia lee el bloque solicitado en el disco, lo escribe en la memoria intermedia y pasa la dirección del bloque en la memoria principal al solicitante. Las acciones internas del gestor de la memoria intermedia resultan transparentes para los programas que formulan solicitudes de bloques de disco.

Si está familiarizado con los conceptos de los sistemas operativos, el lector observará que el gestor de la memoria intermedia no parece ser más que un gestor de memoria virtual, como los que se hallan en la mayor parte de los sistemas operativos. Una diferencia radica en que el tamaño de las bases de datos puede ser mucho mayor que el espacio de direcciones de hardware de la máquina, por lo que las direcciones de memoria no resultan suficientes para direccionar todos los bloques del disco. Además, para dar un buen servicio al sistema de bases de datos, el gestor de la memoria intermedia debe utilizar técnicas más complejas que los esquemas habituales de gestión de la memoria virtual:

- **Estrategia de sustitución.** Cuando no queda espacio libre en la memoria intermedia hay que eliminar un bloque de esta antes de que se pueda leer otro nuevo. La mayor parte de los sistemas operativos utilizan un esquema de **menos recientemente utilizado** (*least recently used*: LRU), en el que se vuelve a escribir en el disco y se elimina de la memoria intermedia el bloque al que se ha hecho referencia menos recientemente. Este sencillo enfoque se puede mejorar para las aplicaciones de bases de datos.
- **Bloques clavados.** Para que el sistema de bases de datos pueda recuperarse de las caídas del sistema (Capítulo 16) hay que restringir las ocasiones en que los bloques se pueden volver a escribir en el disco. Por ejemplo, la mayor parte de los sistemas de recuperación exigen que no se escriban en disco los bloques mientras se esté procediendo a su actualización. Se dice que los bloques que no se permite que se vuelvan a escribir en el disco están **clavados.** Aunque muchos sistemas operativos no permiten trabajar con bloques clavados, esta característica resulta fundamental para los sistemas de bases de datos resistentes a las caídas.
- **Salida forzada de los bloques.** Hay situaciones en las que hace falta volver a escribir los bloques en el disco, aunque no se necesite el espacio de memoria intermedia que ocupan. Este proceso de escritura se denomina **salida forzada** del bloque. Se verá el motivo de las salidas forzadas en el Capítulo 16; en resumen, el contenido de la memoria principal y, por tanto, el de la memoria intermedia se pierden en las caídas, mientras que los datos del disco suelen sobrevivir a ellas.

10.8.2. Políticas para la sustitución de la memoria intermedia

El objetivo de las estrategias de sustitución de los bloques de memoria intermedia es minimizar los accesos al disco. En los programas de propósito general no resulta posible predecir con precisión a qué bloques se hará referencia. Por tanto, el sistema operativo utiliza la pauta anterior de referencias a bloques para predecir las futuras. La suposición que suele hacerse es que es probable que se vuelva a hacer referencia a los bloques a los que se ha hecho referencia recientemente. Por tanto, si hay que sustituir un bloque, se sustituye el bloque al que se ha hecho referencia menos recientemente. Este enfoque se denomina **esquema de sustitución de bloques utilizados menos recientemente (LRU).**

```
for each tupla i de profesor do
   for each tupla d de departamento do
      if i[nombre_dept] = d[nombre_dept]
      then begin
         sea x una tupla definida como:
         x[ID] := i[ID]
         x[nombre_dept] := i[nombre_dept]
         x[nombre] := i[nombre]
         x[sueldo] := i[sueldo]
         x[edificio] := d[edificio]
         x[presupuesto] := d[presupuesto]
         incluya tupla x como parte de profesor ⋈ departamento
      end
   end
end
```

Figura 10.17. Procedimiento para calcular la reunión.

LRU es un esquema de sustitución aceptable para los sistemas operativos. Sin embargo, los sistemas de bases de datos pueden predecir la pauta de referencias futuras con más precisión que los sistemas operativos. Las peticiones de los usuarios al sistema de bases de datos comprenden varias etapas. El sistema de bases de datos suele poder determinar con antelación los bloques que se necesitarán examinando cada una de las etapas necesarias para llevar a cabo la operación solicitada por el usuario. Por tanto, a diferencia de los sistemas operativos, que deben confiar en el pasado para predecir el futuro, los sistemas de bases de datos pueden tener información relativa, al menos, al futuro a corto plazo.

Para ilustrar la manera en que la información sobre el futuro acceso a los bloques permite mejorar la estrategia LRU considere el procesamiento de la consulta SQL:

select *
from *profesor* **natural join** *departamento*;

Suponga que la estrategia escogida para procesar esta solicitud viene dada por el programa en seudocódigo mostrado en la Figura 10.17 (se estudiarán otras estrategias más eficientes en el Capítulo 12).

Suponga que las dos relaciones de este ejemplo se guardan en archivos diferentes. En este ejemplo se puede ver que, una vez que se ha procesado la tupla de *profesor*, no vuelve a ser necesaria. Por tanto, una vez completado el procesamiento de un bloque completo de tuplas de *profesor*, ese bloque ya no se necesita en la memoria principal, aunque se haya utilizado recientemente. Deben darse instrucciones al gestor de la memoria intermedia para que libere el espacio ocupado por el bloque de *profesor* en cuanto se haya procesado la última tupla. Esta estrategia de gestión de la memoria intermedia se denomina estrategia **de extracción inmediata.**

Considere ahora los bloques que contienen las tuplas de *departamento*. Hay que examinar cada bloque de las tuplas *departamento* una vez por cada tupla de la relación *profesor*. Cuando se completa el procesamiento del bloque *departamento*, se sabe que no se accederá nuevamente a él hasta que no se hayan procesado todos los demás bloques de *departamento*. Por tanto, el bloque de *departamento* al que se haya hecho referencia más recientemente será el último bloque al que se vuelva a hacer referencia, y el bloque de *departamento* al que se haya hecho referencia menos recientemente será el bloque al que se vuelva a hacer referencia a continuación. Este conjunto de suposiciones es justo el contrario del que forma la base de la estrategia LRU. En realidad, la estrategia óptima de sustitución de bloques para el procedimiento anterior es la **estrategia más recientemente utilizada** (*most recently used*: MRU). Si hay que eliminar de la memoria intermedia un bloque de *departamento*, la estrategia MRU escoge el bloque utilizado más recientemente (los bloques en uso no se pueden eliminar).

Para que la estrategia MRU funcione correctamente en el ejemplo propuesto, el sistema debe clavar el bloque de *departamento* que se esté procesando. Después de que se haya procesado la última tupla de *departamento*, el bloque se desclava y pasa a ser el bloque utilizado más recientemente.

Además de utilizar la información que pueda tener el sistema respecto de la solicitud que se esté procesando, el gestor de la memoria intermedia puede utilizar información estadística relativa a la probabilidad de que una solicitud haga referencia a una relación concreta. Por ejemplo, el diccionario de datos (como se trató en la Sección 10.7) que realiza un seguimiento del esquema lógico de las relaciones y de la información de su almacenamiento físico es una de las partes de la base de datos a la que se tiene acceso con mayor frecuencia. Por tanto, el gestor de la memoria intermedia debe intentar no eliminar de la memoria principal los bloques del diccionario de datos, a menos que se vea obligado a hacerlo. En el Capítulo 11 se estudian los índices de los archivos. Dado que puede que se acceda más frecuentemente al índice del archivo que al propio archivo, el gestor de la memoria intermedia no deberá, en general, eliminar de la memoria principal los bloques del índice si se dispone de alternativas.

La estrategia ideal para la sustitución de bloques necesita información sobre las operaciones de la base de datos, las que se estén realizando y las que se vayan a realizar en el futuro. No se conoce una sola estrategia que sea adecuada para todas las situaciones posibles. En realidad, un número sorprendentemente grande de bases de datos utiliza LRU, a pesar de sus defectos. Las preguntas prácticas y los ejercicios exploran estrategias alternativas.

La estrategia utilizada por el gestor de la memoria intermedia para la sustitución de los bloques se ve influida por factores distintos del momento en que se volverá a hacer referencia a cada bloque. Si el sistema está procesando de manera concurrente las solicitudes de varios usuarios, puede que el subsistema para el control de concurrencia (Capítulo 15) tenga que posponer ciertas solicitudes para asegurar la conservación de la consistencia de la base de datos. Si se proporciona al gestor de la memoria intermedia información del subsistema de control de concurrencia que indique las solicitudes que se posponen, puede utilizar esa información para modificar su estrategia de sustitución de los bloques. Concretamente, los bloques que necesiten las solicitudes activas (no pospuestas) pueden conservarse en la memoria intermedia a expensas de los que necesiten las solicitudes pospuestas.

El subsistema para la recuperación de caídas (Capítulo 16) impone severas restricciones a la sustitución de bloques. Si se ha modificado un bloque, no se permite que el gestor de la memoria intermedia vuelva a copiar al disco la versión nueva del bloque existente en la memoria intermedia, dado que eso destruiría la versión anterior. Por el contrario, el gestor de bloques debe solicitar permiso del subsistema para la recuperación de caídas antes de escribir cada bloque. Puede que el subsistema para la recuperación de caídas exija que se fuerce la salida de otros bloques antes de conceder autorización al gestor de la memoria intermedia para que escriba el bloque solicitado. En el Capítulo 16 se define con precisión la interacción entre el gestor de la memoria intermedia y el subsistema para la recuperación de caídas.

10.9. Resumen

- En la mayor parte de los sistemas informáticos hay varios tipos de almacenamiento de datos. Estos sistemas se clasifican según la velocidad con la que se puede acceder a los datos, el coste de adquisición de la memoria por unidad de datos y su fiabilidad. Entre los medios disponibles figuran la memoria caché, la principal, la flash, los discos magnéticos, los discos ópticos y las cintas magnéticas.
- La fiabilidad de los medios de almacenamiento la determinan dos factores: si un corte en el suministro eléctrico o una caída del sistema hace que los datos se pierdan y la probabilidad de fallo físico del dispositivo de almacenamiento.
- Se puede reducir la probabilidad de fallo físico conservando varias copias de los datos. Para los discos se pueden crear imágenes. También se pueden usar métodos más sofisticados como las disposiciones redundantes de discos independientes (RAID). Mediante la distribución de los datos entre los discos estos métodos ofrecen elevadas tasas de transferencia en los accesos de gran tamaño; la introducción de la redundancia entre los discos mejora mucho la fiabilidad. Se han propuesto varias organizaciones RAID diferentes, con características de coste, rendimiento y fiabilidad diversas. Las organizaciones RAID de nivel 1 (creación de imágenes) y de nivel 5 son las más utilizadas.
- Los archivos se pueden organizar lógicamente como secuencias de registros asignados a bloques de disco. Un enfoque de la asociación de la base de datos con los archivos es utilizar varios archivos y guardar registros de una única longitud fija en cualquier archivo dado. Una alternativa es estructurar los archivos de modo que puedan aceptar registros de longitud variable. El método de la página con ranuras se usa mucho para manejar los registros de longitud variable en los bloques de disco.
- Dado que los datos se transfieren entre el almacenamiento en disco y la memoria principal en unidades de bloques, merece la pena asignar los registros de los archivos a los bloques de modo que cada bloque contenga registros relacionados entre sí. Si se puede tener acceso a varios de los registros deseados utilizando solo un acceso a bloques, se evitan accesos al disco. Dado que los accesos al disco suelen ser el cuello de botella del rendimiento de los sistemas de bases de datos, una cuidadosa asignación de registros a los bloques puede ofrecer mejoras significativas del rendimiento.
- El diccionario de datos, también denominado catálogo del sistema, guarda información de los metadatos, que son datos relativos a los datos, como el nombre de las relaciones, el nombre de los atributos y su tipo, la información de almacenamiento, las restricciones de integridad y la información de usuario.
- Una manera de reducir el número de accesos al disco es conservar todos los bloques posibles en la memoria principal. Dado que allí no se pueden guardar todos, hay que gestionar la asignación del espacio disponible en la memoria principal para el almacenamiento de los bloques. La *memoria intermedia (buffer)* es la parte de la memoria principal disponible para el almacenamiento de las copias de los bloques del disco. El subsistema responsable de la asignación del espacio de la memoria intermedia se denomina *gestor de la memoria intermedia*.

Términos de repaso

- Medios de almacenamiento físico.
  - Caché.
  - Memoria principal.
  - Memoria flash .
  - Disco magnético.
  - Almacenamiento óptico.
- Disco magnético.
  - Plato.
  - Disco duro.
  - Disquetes.
  - Pistas.
  - Sectores.
  - Cabeza de lectura y escritura.
  - Brazo del disco.
  - Cilindro.
  - Controlador de discos.
  - Suma de comprobación.
  - Reasignación de sectores defectuosos.
- Medidas de rendimiento de los discos.
  - Tiempo de acceso.
  - Tiempo de búsqueda.
  - Latencia rotacional.
  - Velocidad de transferencia de datos.
  - Tiempo medio entre fallos (MTTF).
- Bloque de disco.
- Optimización del acceso a bloques de disco.
  - Planificación del brazo.
  - Algoritmo del ascensor.
  - Organización de archivos.
  - Desfragmentación.
  - Memorias intermedias de escritura no volátiles.
  - Memoria no volátil de acceso aleatorio (NVRAM).
  - Disco del registro histórico.
- Disposición redundante de discos independientes (RAID).
  - Creación de imágenes.
  - Distribución de datos.
  - Distribución en el nivel de bits.
  - Distribución en el nivel de bloques.
- Niveles de RAID.
  - Nivel 0 (distribución de bloques, sin redundancia).
  - Nivel 1 (distribución de bloques, con creación de imágenes).
  - Nivel 3 (distribución de bits, con paridad).
  - Nivel 5 (distribución de bloques, con paridad distribuida).
  - Nivel 6 (distribución de bloques, con redundancia P+Q).
- Rendimiento de la reconstrucción.
- RAID software.
- RAID hardware.
- Intercambio en caliente.
- Almacenamiento terciario.
  - Discos ópticos.
  - Cintas magnéticas.
  - Cambiadores automáticos.
- Archivo.
- Organización de archivos.
  - Cabecera del archivo.
  - Lista libre.

- Registros de longitud variable.
  - Estructura de páginas con ranuras.
- Objetos de gran tamaño.
- Organización de archivos en montículos.
- Organización secuencial de archivos.
- Organización asociativa (*hash*) de archivos.
- Organización de archivos en agrupaciones de varias tablas.
- Clave de búsqueda.
- Diccionario de datos.
- Catálogo del sistema.
- Memoria intermedia (*buffer*).
  - Gestor de la memoria intermedia.
  - Bloques clavados.
  - Salida forzada de bloques.
- Políticas de sustitución de la memoria intermedia.
  - Menos recientemente utilizada (*least recently used*: LRU).
  - Extracción inmediata.
  - Más recientemente utilizada (*most recently used*: MRU).

Ejercicios prácticos

10.1. Considere la disposición de los bloques de datos y de paridad de cuatro discos de la Figura 10.18: B_i representa los bloques de datos; P_i, los bloques de paridad. El bloque de paridad P_i es el bloque de paridad para los bloques de datos B_{4i-3} a B_{4i}. Indique los problemas que puede presentar esta disposición, si los hay.

10.2. Con respecto al almacenamiento flash:

a. ¿Cómo es la tabla de traducción de las memorias flash que se usa para asociar los números de páginas lógicas en números de páginas físicas?

b. Suponga que tiene un sistema de almacenamiento flash de 60 gigabytes, con un tamaño de página de 4096 bytes. ¿Cuál será el tamaño de la tabla de traducciones, suponiendo que cada página es de 32 bits de direcciones y la tabla se guarda como un array?

c. Sugiera cómo reducir el tamaño de la tabla de traducción si normalmente se suelen asociar grandes intervalos de números de página lógicos consecutivos a números de página físicos consecutivos.

Disco 1	Disco 2	Disco 3	Disco 4
B_1	B_2	B_3	B_4
P_1	B_5	B_6	B_7
B_8	P_2	B_9	B_{10}
...	...	...	...

Figura 10.18. Organización de bloques de paridad y de datos.

10.3. Un fallo en el suministro eléctrico que se produzca mientras se escribe un bloque del disco puede dar lugar a que el bloque solo se escriba parcialmente. Suponga que se pueden detectar los bloques escritos parcialmente. Un proceso atómico de escritura de bloque es aquel en el que se escribe el bloque entero o no se escribe en absoluto (es decir, no hay procesos de escritura parciales). Propónganse esquemas para conseguir el efecto de los procesos atómicos de escritura de bloques con los siguientes esquemas RAID. Los esquemas deben implicar procesos de recuperación de fallos.

a. RAID de nivel 1 (creación de imágenes).

b. RAID de nivel 5 (entrelazado de bloques, paridad distribuida).

10.4. Considere el borrado del registro 5 del archivo de la Figura 10.6. Compare las ventajas relativas de las siguientes técnicas para implementar el borrado:

a. Trasladar el registro 6 al espacio que ocupaba el registro 5 y desplazar el registro 7 al espacio ocupado por el registro 6.

b. Trasladar el registro 7 al espacio que ocupaba el registro 5.

c. Marcar el registro 5 como borrado y no desplazar ningún registro.

10.5. Indique la estructura del archivo de la Figura 10.7 después de cada uno de los pasos siguientes:

a. Insertar (24556, Turnamian, Finanzas, 98000).

b. Borrar el registro 2.

c. Insertar (34556, Thompson, Música, 67000).

10.6. Considere las relaciones *sección* y *matricula*. Indique un ejemplar de estas dos relaciones, con tres secciones, cada una con cinco estudiantes. Indique una estructura de archivo para estas relaciones que usa organización de archivos en agrupaciones de varias tablas.

10.7. Considere la siguiente técnica de mapa de bits para realizar el seguimiento del espacio libre de un archivo. Por cada bloque del archivo se mantienen dos bits en el mapa. Si el bloque está lleno entre el 0 y el 30 por ciento, los bits son 00; entre el 30 y el 60 por ciento, 01; entre el 60 y el 90 por ciento, 10 y, por encima del 90 por ciento, 11. Estos mapas de bits se pueden mantener en la memoria incluso para archivos de gran tamaño.

a. Describa la manera de mantener actualizado el mapa de bits mientras se insertan y se eliminan registros.

b. Describa la ventaja de la técnica de los mapas de bits frente a las listas libres en la búsqueda de espacio libre y en la actualización de la información.

10.8. Resulta importante ser capaz de descubrir rápidamente si un bloque se encuentra en la memoria intermedia y, si es así, dónde se encuentra. Dado que los tamaños de las memorias intermedias de las bases de datos son muy grandes, ¿qué estructura (en memoria) se debería usar para ello?

10.9. Indique un ejemplo de expresión de álgebra relacional y de estrategia de procesamiento de consultas en cada una de las situaciones siguientes:

a. MRU es preferible a LRU.

b. LRU es preferible a MRU.

Ejercicios

10.10. Indique los medios de almacenamiento físico disponibles en las computadoras que utiliza habitualmente. Indique la velocidad con la que se puede acceder a los datos en cada medio.

10.11. ¿Cómo afecta la reasignación por los controladores de disco de los sectores dañados a la velocidad de recuperación de los datos?

10.12. Los sistemas RAID suelen permitir la sustitución de los discos averiados sin interrupción del acceso al sistema. Por tanto, los datos del disco averiado deben reconstruirse y escribirse en el disco de repuesto mientras el sistema se halla en funcionamiento. ¿Con cuál de los niveles RAID es menor la interferencia entre la reconstrucción y el acceso a los discos? Explique la respuesta.

10.13. ¿Qué es el rastreo (*scrubbing*), en el contexto de los sistemas RAID, y por qué es importante?

10.14. En la representación de registros de longitud variable se usa un mapa de bits de nulos para indicar que un atributo tiene el valor nulo.

- **a.** Para los campos de longitud variable, si el valor es nulo, ¿qué debería guardar en el desplazamiento de los campos de desplazamiento y de longitud?
- **b.** En algunas aplicaciones, las tuplas tienen un gran número de atributos, la mayoría de los cuales valen nulo. ¿Puede modificar la representación de los registros, de forma que la única sobrecarga de un atributo nulo sea un único bit en el mapa de bits de nulos?

10.15. Explique por qué la asignación de registros a los bloques afecta de manera significativa al rendimiento de los sistemas de bases de datos.

10.16. Si es posible, determine la estrategia de gestión de la memoria intermedia del sistema operativo que se ejecuta en su computadora y los mecanismos de que dispone para controlar la sustitución de páginas. Explique el modo en que el control sobre la sustitución que proporciona pudiera ser útil para la implementación de sistemas de bases de datos.

10.17. Indique dos ventajas y dos inconvenientes de cada una de las estrategias siguientes para el almacenamiento de bases de datos relacionales:

- **a.** Guardar cada relación en un archivo.
- **b.** Guardar varias relaciones (quizá toda la base de datos) en un archivo.

10.18. En la organización secuencial de los archivos, ¿por qué se utiliza un bloque de desbordamiento aunque solo haya, en ese momento, un registro de desbordamiento?

10.19. Cree una versión normalizada de la relación *Metadatos_índices* y explique por qué el empleo de la versión normalizada supondría una pérdida de rendimiento.

10.20. Si se tienen datos que no se deben perder en un fallo de disco y la escritura de datos es intensiva, ¿cómo almacenaría esos datos?

10.21. En discos de generaciones anteriores el número de sectores por pista era el mismo en todas las pistas. Los discos de las generaciones actuales tienen más sectores por pista en las externas y menos en las internas (ya que son más cortas). ¿Cuál es el efecto de este cambio en cada uno de los tres indicadores principales de la velocidad de los discos?

10.22. Los gestores habituales de la memoria intermedia dan por supuesto que cada página es del mismo tamaño y cuesta lo mismo leerla. Considere un gestor que, en lugar de LRU, utilice la tasa de referencia a objetos, es decir, la frecuencia con que se ha accedido a ese objeto en los últimos n segundos. Suponga que se desea almacenar en la memoria intermedia objetos de distinto tamaño y con costes de lectura diferentes (como las páginas web, cuyo coste de lectura depende del sitio desde el que se busquen). Sugiera cómo puede elegir el gestor la página que debe descartar de la memoria intermedia.

Notas bibliográficas

Hennessy et ál. [2006] es un popular libro de texto de arquitectura de computadoras, que incluye los aspectos de hardware de la memoria intermedia con traducción anticipada, de las cachés y de las unidades de gestión de la memoria. Rosch [2003] presenta una excelente visión general del hardware de las computadoras, incluido un extenso tratamiento de todos los tipos de tecnologías de almacenamiento, como los disquetes, los discos magnéticos, los discos ópticos, las cintas y las interfaces de almacenamiento. Patterson [2004] ofrece un buen estudio sobre el modo en que la mejora de la latencia ha ido por detrás de la mejora del ancho de banda (velocidad de transferencia).

Con el rápido crecimiento de las velocidades de la CPU, las memorias caché ubicadas en la CPU han llegado a ser mucho más rápidas que la memoria principal. Aunque los sistemas de bases de datos no controlan lo que se almacena en la caché, cada vez hay más motivos para organizar los datos en memoria y escribir los programas de forma que se maximice el empleo de la caché. Entre los trabajos en esta área figuran Rao y Ross [2000], Ailamaki et ál. [2001] Zhou y Ross [2004], García y Korth [2005] y Cieslewicz et ál. [2009].

Las especificaciones de las unidades de disco actuales se pueden obtener de los sitios web de sus fabricantes, como Hitachi, LaCie, Iomega, Seagate, Maxtor y Western Digital.

Las disposiciones redundantes de discos independientes (RAID) se explican en Patterson et ál. [1988]. Chen et ál. [1994] presentan una excelente revisión de los principios y de la aplicación de RAID. Los códigos de Reed-Solomon se tratan en Pless [1998].

El uso de memorias intermedias de datos en sistemas móviles se trata en Imielinski y Badrinath [1994], Imielinski y Korth [1996] y Chandrasekaran et ál. [2003].

La estructura de almacenamiento de sistemas concretos de bases de datos, como DB2 de IBM, Oracle o SQL Server de Microsoft y PostgreSQL se documentan en sus respectivos manuales de sistema.

La gestión de las memorias intermedias se explica en la mayor parte de los textos sobre sistemas operativos, incluido Silberschatz et ál. [2008]. Chou y Dewitt [1985] presentan algoritmos para la gestión de la memoria intermedia en los sistemas de bases de datos y describen un método de medida del rendimiento.

Indexación y asociación 11

Muchas consultas solo hacen referencia a una pequeña parte de los registros de un archivo. Por ejemplo, la consulta «Buscar a todos los profesores del departamento de Física» o «Buscar el número total de créditos que ha conseguido el estudiante con *ID* 22201» hace referencia solamente a una fracción de los registros de estudiantes. No es eficiente que el sistema tenga que leer todos los registros de la relación *profesor* para comprobar si el valor del campo *nombre_dept* es «Física». Y no es eficiente que el sistema tenga que leer todas las relaciones *estudiante* para encontrar las tuplas con *ID* «22201». Lo más adecuado sería que el sistema fuese capaz de localizar directamente esos registros. Para facilitar estas formas de acceso se diseñan estructuras adicionales que se asocian con los archivos.

11.1. Conceptos básicos

Un índice para un archivo del sistema de bases de datos funciona como el índice de este libro. Si se va a buscar un tema (especificado por una palabra o una frase) se puede buscar en el índice al final del libro, encontrar las páginas en las que aparece y después leerlas para encontrar la información buscada. Las palabras del índice están ordenadas alfabéticamente, lo cual facilita la búsqueda. Además, el índice es mucho más pequeño que el libro, con lo que se reduce aún más el esfuerzo necesario para encontrar las palabras en cuestión.

Los índices de los sistemas de bases de datos juegan el mismo papel que los índices de los libros en las bibliotecas. Por ejemplo, para recuperar un registro *estudiante* dado su *ID*, el sistema de bases de datos buscaría en un índice para encontrar el bloque de disco en el que se localice el registro correspondiente, y entonces extraería ese bloque de disco para obtener el registro *estudiante* correspondiente.

Almacenar una lista ordenada de los *ID* de los estudiantes no funcionaría bien en bases de datos muy grandes con miles de estudiantes, ya que el propio índice sería muy grande; más aún, incluso aunque mantener ordenado el índice reduce el tiempo de búsqueda, encontrar a un estudiante puede consumir mucho tiempo. En su lugar se usan técnicas más sofisticadas de indexación. Algunas de estas técnicas se estudiarán más adelante.

Hay dos tipos básicos de índices:

- **Índices ordenados.** Estos índices están basados en una disposición ordenada de los valores.
- **Índices asociativos.** Estos índices están basados en una distribución uniforme de los valores a través de una serie de cajones *(buckets)*. El valor asignado a cada cajón está determinado por una función, llamada *función de asociación (hash function)*.

Se considerarán varias técnicas de indexación ordenada y de asociación. Ninguna de ellas es la mejor. Sin embargo, para cada aplicación específica de bases de datos existe una técnica más apropiada. Hay que valorar cada una de ellas según los siguientes criterios:

- **Tipos de acceso:** los tipos de acceso que se soportan eficientemente. Estos tipos podrían incluir la búsqueda de registros con un valor concreto en un atributo, o la búsqueda de los registros cuyos atributos contengan valores en un intervalo especificado.
- **Tiempo de acceso:** el tiempo que se tarda en hallar un determinado elemento de datos, o conjunto de elementos, usando la técnica en cuestión.
- **Tiempo de inserción:** el tiempo empleado en insertar un nuevo elemento de datos. Este valor incluye el tiempo utilizado en hallar el lugar apropiado donde insertar el nuevo elemento de datos, así como el tiempo empleado en actualizar la estructura del índice.
- **Tiempo de borrado:** el tiempo empleado en borrar un elemento de datos. Este valor incluye el tiempo utilizado en hallar el elemento a borrar, así como el tiempo empleado en actualizar la estructura del índice.
- **Espacio adicional requerido:** el espacio adicional ocupado por la estructura del índice. Como normalmente la cantidad necesaria de espacio adicional suele ser moderada, es razonable sacrificar el espacio para alcanzar un mejor rendimiento.

A menudo se desea tener más de un índice por archivo. Por ejemplo, se puede buscar un libro según el autor, la materia o el título.

Los atributos o conjunto de atributos usados para buscar en un archivo se llaman **clave de búsqueda.** Hay que observar que esta definición de *clave* difiere de la usada en *clave primaria*, *clave candidata* y *superclave*. Este doble significado de *clave* está (desafortunadamente) muy extendido en la práctica. Usando el concepto de clave de búsqueda se determina que, si existen varios índices para un archivo, existirán varias claves de búsqueda.

11.2. Índices ordenados

Para permitir un acceso aleatorio rápido a los registros de un archivo se puede usar una estructura de índice. Cada estructura de índice está asociada con una clave de búsqueda concreta. Al igual que en el catálogo de una biblioteca, un índice almacena de manera ordenada los valores de las claves de búsqueda, y asocia a cada clave los registros que contienen esa clave de búsqueda.

Los registros en el archivo indexado pueden estar a su vez almacenados siguiendo un orden, semejante a como los libros están ordenados en una biblioteca por algún atributo como el número decimal Dewey. Un archivo puede tener varios índices según diferentes claves de búsqueda. Si el archivo que contiene los registros está ordenado secuencialmente, el índice cuya clave de búsqueda especifica el orden secuencial del archivo es el **índice con agrupación** (*clustering index*). Los índices con agrupación también se llaman **índices primarios;** el término índice primario se usa algunas veces para hacer alusión a un índice según una clave primaria pero en realidad se puede usar sobre cualquier clave de búsqueda. La clave de búsqueda de un índice con agrupación es normalmente la clave primaria, aunque no necesariamente. Los índices cuyas claves de búsqueda especifican un orden diferente del orden secuencial del archivo se llaman **índices sin agrupación** (*non clustering indices*) o **secundarios.** Se suelen usar los términos «agrupado» y «no agrupado» en lugar de «con agrupación» y «sin agrupación».

10101	Srinivasan	Informática	65000	
12121	Wu	Finanzas	90000	
15151	Mozart	Música	40000	
22222	Einstein	Física	95000	
32343	El Said	Historia	60000	
33456	Gold	Física	87000	
45565	Katz	Informática	75000	
58583	Califieri	Historia	62000	
76543	Singh	Finanzas	80000	
76766	Crick	Biología	72000	
83821	Brandt	Informática	92000	
98345	Kim	Electrónica	80000	

Figura 11.1. Archivo secuencial para los registros de *profesor*.

En las Secciones 11.2.1 hasta la 11.2.3 se asume que todos los archivos están ordenados secuencialmente según alguna clave de búsqueda. Estos archivos con un índice con agrupación según la clave de búsqueda se llaman **archivos secuenciales indexados.** Representan uno de los esquemas de índices más antiguos usados por los sistemas de bases de datos. Se usan en aquellas aplicaciones que demandan un procesamiento secuencial del archivo completo, así como un acceso directo a sus registros. En la Sección 11.2.4 se tratan los índices secundarios.

En la Figura 11.1 se muestra un archivo secuencial de los registros *profesor* tomados del ejemplo de la universidad. En esta figura, los registros están almacenados según el orden del *ID* del profesor, que se usa como clave de búsqueda.

11.2.1. Índices densos y dispersos

Un **registro índice** o **entrada del índice** consiste en un valor de la clave de búsqueda y punteros a uno o más registros con ese valor de la clave de búsqueda. El puntero a un registro consiste en el identificador de un bloque de disco y un desplazamiento en el bloque de disco para identificar el registro dentro del bloque.

Existen dos clases de índices ordenados que se pueden usar:

- **Índice denso:** en un índice denso, aparece un registro índice por cada valor de la clave de búsqueda en el archivo. En un índice denso con agrupación el registro índice contiene el valor de la clave y un puntero al primer registro con ese valor de la clave de búsqueda. El resto de registros con el mismo valor de la clave de búsqueda se almacenan consecutivamente tras el primer registro; dado que el índice es con agrupación, los registros se ordenan sobre la misma clave de búsqueda.

  En un índice sin agrupación denso, los índices deben almacenar una lista de punteros a todos los registros con el mismo valor de la clave de búsqueda.

10101	10101	Srinivasan	Informática	65000	
12121	12121	Wu	Finanzas	90000	
15151	15151	Mozart	Música	40000	
22222	22222	Einstein	Física	95000	
32343	32343	El Said	Historia	60000	
33456	33456	Gold	Física	87000	
45565	45565	Katz	Informática	75000	
58583	58583	Califieri	Historia	62000	
76543	76543	Singh	Finanzas	80000	
76766	76766	Crick	Biología	72000	
83821	83821	Brandt	Informática	92000	
98345	98345	Kim	Electrónica	80000	

Figura 11.2. Índice denso.

- **Índice disperso:** en un índice disperso, solo aparece un registro índice para algunos de los valores de la clave de búsqueda. Los índices dispersos solo se pueden usar si la relación se almacena ordenada según la clave de búsqueda, es decir, el índice es un índice de agrupamiento. Al igual que en los índices densos, cada registro índice contiene un valor de la clave de búsqueda y un puntero al primer registro con ese valor de la clave. Para localizar un registro se busca la entrada del índice con el valor más grande que sea menor o igual al valor que se está buscando. Se empieza por el registro apuntado por esa entrada del índice y se continúa con los punteros del archivo hasta encontrar el registro deseado.

Las Figuras 11.2 y 11.3 son ejemplos de índices densos y dispersos, respectivamente, para el archivo *profesor*. Suponga que se desea buscar los registros del profesor con *ID* «22222». Mediante el índice denso de la Figura 11.2, se sigue el puntero que va directo al registro deseado. Como *ID* es una clave primaria, solo existe un único registro y la búsqueda ha terminado. Si se usa el índice disperso (Figura 11.3), no se encontraría entrada del índice para «22222». Como la última entrada (en orden numérico) antes de «22222» es «10101», se sigue ese puntero. Entonces se lee el archivo *profesor* en orden secuencial hasta encontrar el registro deseado.

Suponga un diccionario impreso. En la cabecera de las páginas aparece la primera palabra de la página. Las palabras que se encuentran en la cabecera forman juntas un índice disperso del contenido de las páginas del diccionario.

Otro ejemplo: suponga que el valor de la clave de búsqueda no es una clave primaria. La Figura 11.4 muestra un índice de agrupamiento denso del archivo *profesor* en el que la clave de búsqueda es *nombre_dept*. Observe que en este caso el archivo *profesor* está ordenado por la clave de búsqueda *nombre_dept*, en lugar de por *ID*, sino el índice por *nombre_dept* sería un índice sin agrupamiento. Suponga que se buscan los registros del departamento de Historia. Usando el índice denso de la Figura 11.4 se sigue el puntero directamente al primer registro de Historia. Se procesa este registro y se sigue el puntero de ese registro para localizar el siguiente registro en el orden de la clave de búsqueda *(nombre_dept)*. Se continúan procesando registros hasta que se encuentre un registro de un departamento que no sea el de Historia.

Como se ha visto, generalmente es más rápido localizar un registro si se usa un índice denso en vez de un índice disperso. Sin embargo, los índices dispersos tienen algunas ventajas sobre los índices densos, como el utilizar un espacio más reducido y un mantenimiento adicional menor para las inserciones y borrados.

Existe un compromiso que el diseñador del sistema debe mantener entre el tiempo de acceso y el espacio adicional requerido. Aunque la decisión sobre este compromiso depende de la aplicación en particular, un buen compromiso es tener un índice disperso con una entrada del índice por cada bloque. La razón por la cual este diseño alcanza un buen compromiso reside en que el mayor coste de procesar una petición en la base de datos es el empleado en traer un bloque de disco a la memoria. Una vez el bloque se encuentra en la memoria, el tiempo utilizado en examinarlo es prácticamente inapreciable. Usando este índice disperso se localiza el bloque que contiene el registro solicitado. De este manera, a menos que el registro esté en un bloque de desbordamiento (véase la Sección 10.6.1) se minimizan los accesos a bloques mientras el tamaño del índice se mantiene (y así, el espacio adicional requerido) tan pequeño como sea posible.

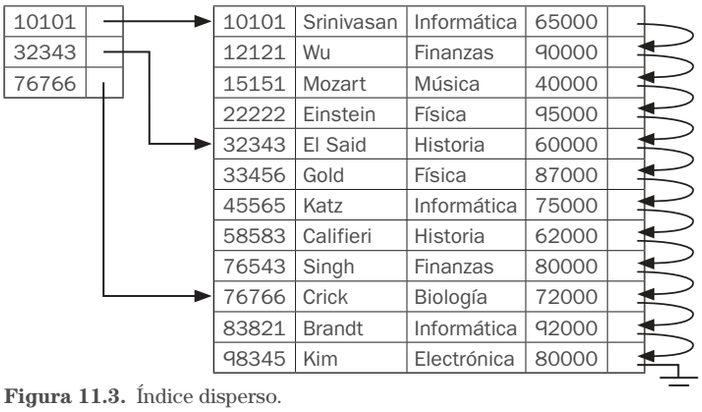


Figura 11.3. Índice disperso.

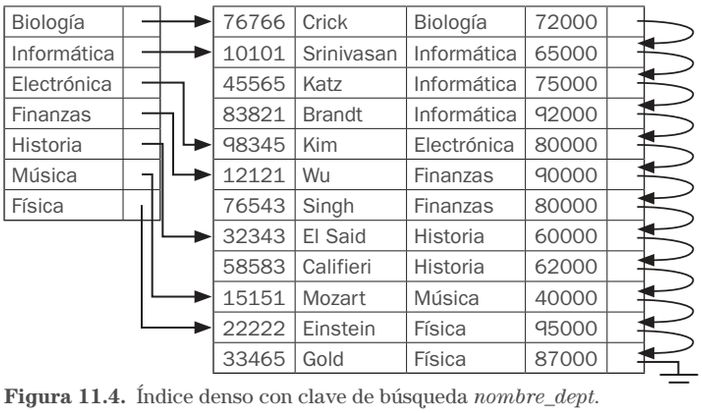


Figura 11.4. Índice denso con clave de búsqueda *nombre_dept*.

Para generalizar la técnica anterior hay que tener en cuenta si los registros de una clave de búsqueda ocupan varios bloques. Es fácil modificar el esquema para acomodarse a esta situación.

11.2.2. Índices multinivel

Suponga que se construye un índice denso de una relación con 1.000.000 de tuplas. Las entradas de los índices son más pequeñas que los registros de datos, por lo que se puede suponer que caben 100 registros índices en un bloque de 4 kilobytes. Por tanto, ocuparía 10.000 bloques. Si la relación tuviese 100.000.000 de tuplas, el índice ocuparía 1.000.000 de bloques o 4 gigabytes de espacio. Los índices de este tamaño se guardan en disco como archivos secuenciales.

Si un índice es lo bastante pequeño como para guardarlo en la memoria principal, el tiempo de búsqueda para encontrar una entrada será reducido. Sin embargo, si el índice es tan grande que se debe guardar en disco, se deben leer los bloques de índice de disco cuando se necesiten. (Incluso si un índice es pequeño para caber en la memoria principal, esta memoria también se necesita para otras tareas, por lo que puede no ser posible mantener el índice completo en la memoria). La búsqueda de una entrada del índice requiere de la lectura de varios bloques de disco.

Se puede usar una búsqueda binaria en el archivo de índice para localizar una entrada, pero la búsqueda sigue teniendo un cierto coste. Si el índice ocupase b bloques, la búsqueda binaria requiere leer hasta $\lceil \log_2(b) \rceil$ bloques ($\lceil x \rceil$ significa el menor entero que es mayor o igual que x; es decir, se redondea siempre hacia arriba). Para un índice de 10.000 bloques, la búsqueda binaria puede necesitar leer hasta 14 bloques. En un sistema de discos en el que la lectura de un bloque tarda un promedio de 10 milisegundos, la búsqueda del índice tardaría 140 milisegundos. Puede no parecer mucho, pero significa que solo se pueden realizar 7 búsquedas por segundo, mientras que mecanismos más eficientes podrían proporcionarnos muchas más búsquedas por segundo, como se verá en breve. Fíjese que, si se han usado bloques de desbordamiento, la búsqueda binaria no es posible. En este caso, normalmente se usa una búsqueda secuencial, que requiere leer b bloques, lo que puede tardar mucho más. Por tanto, el proceso de buscar en un índice grande puede ser muy costoso.

Para resolver este problema el índice se trata como si fuese un archivo secuencial y se construye un índice externo disperso sobre el índice original, que ahora llamaremos índice interno, como se muestra en la Figura 11.5. Fíjese que las entradas del índice siempre están ordenadas, lo que permite que el índice externo sea disperso. Para localizar un registro se usa en primer lugar una búsqueda binaria sobre el índice más externo para buscar el registro con el mayor valor de la clave de búsqueda que sea menor o igual al valor deseado. El puntero apunta a un bloque en el índice más interno. Hay que examinar este bloque hasta encontrar el registro con el mayor valor de la clave que sea menor o igual que el valor deseado. El puntero de este registro apunta al bloque del archivo que contiene el registro buscado.

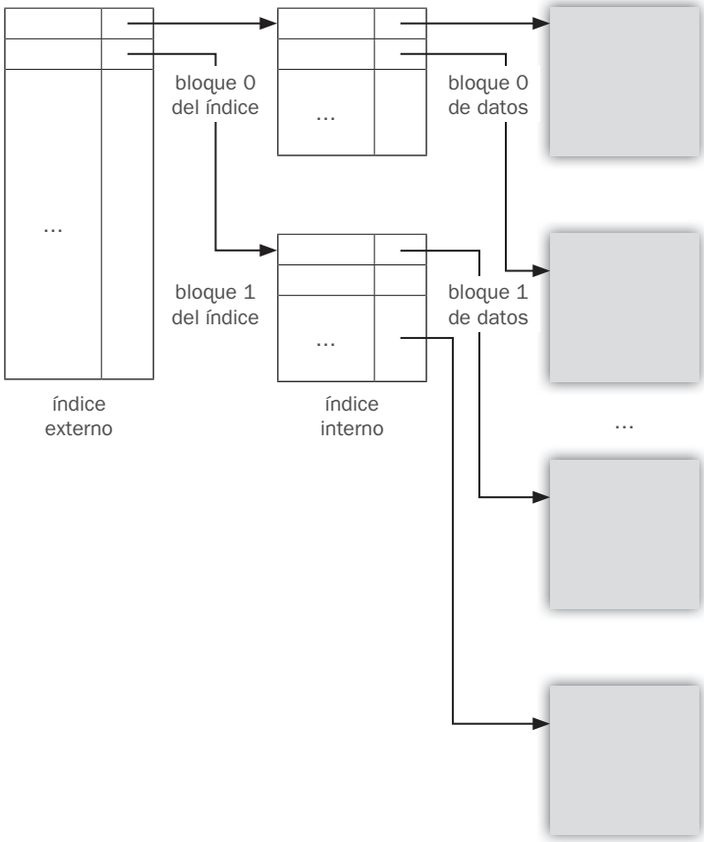


Figura 11.5. Índice disperso de dos niveles.

En nuestro ejemplo, un índice interno con 10.000 bloques requiere 10.000 entradas en el índice externo que ocuparía solo 100 bloques. Si suponemos que el índice externo ya está en la memoria principal, para la búsqueda multinivel se leería solo un bloque de índices para una búsqueda, en lugar de los 14 bloques que se leían con la búsqueda binaria. El resultado es que se pueden realizar 14 veces más búsquedas de índice por segundo.

Si el archivo fuese extremadamente grande, incluso el índice externo puede crecer lo suficiente como para que no quepa en la memoria principal. Con una relación de 100.000.000 de tuplas, el índice interior ocuparía 1.000.000 de bloques y el índice exterior ocuparía 10.000 bloques, o 40 megabytes. Como hay mucha demanda de memoria principal, puede que no sea posible reservar tanta memoria solo para el índice externo. En estos casos, se crea otro nivel más de índices. De hecho se puede repetir este proceso tantas veces como sea necesario. Los índices con dos o más niveles se llaman índices **multinivel.** La búsqueda de registros usando un índice multinivel necesita claramente menos operaciones de E/S que las que se emplean en la búsqueda de registros con la búsqueda binaria.[1]

Los índices multinivel están estrechamente relacionados con la estructura de árbol, tales como los árboles binarios usados para la indexación en memoria. Esta relación se examinará posteriormente en la Sección 11.3.

11.2.3. Actualización del índice

Sin importar el tipo de índice que se esté usando, los índices se deben actualizar siempre que se inserte o borre un registro del archivo. En el caso de que se actualice un registro en el archivo, todos los índices cuyo atributo de la clave de búsqueda se vea afectado por la actualización, también deben ser actualizados; por ejemplo, si el departamento de un profesor cambia, hay que actualizar de forma correspondiente un índice sobre el atributo *nombre_dept* de *profesor*. Esta actualización de un registro se puede modelar como un borrado del registro antiguo, seguido de una inserción del nuevo valor del registro, lo que genera un borrado de índice seguido de una inserción de índice. Como resultado, solo se necesita considerar las inserciones y borrados del índice, no se necesitan explícitamente las actualizaciones.

En primer lugar se describirán los algoritmos para actualizar índices de un solo nivel.

- **Inserción.** Primero se realiza una búsqueda usando el valor de la clave de búsqueda del registro a insertar. Las acciones que emprende el sistema a continuación dependen de si el índice es denso o disperso.
  - Índices densos:
    1. Si el valor de la clave de búsqueda no aparece en el índice, el sistema inserta en este un registro índice con el valor de la clave de búsqueda en la posición adecuada.
    2. En caso contrario se emprenden las siguientes acciones:
       a. Si el registro índice almacena punteros a todos los registros con el mismo valor de la clave de búsqueda, el sistema añade un puntero al nuevo registro en el registro índice.
       b. En caso contrario, el registro índice almacena un puntero solo hacia el primer registro con el valor de la clave de búsqueda. El sistema sitúa el registro insertado después de los otros con los mismos valores de la clave de búsqueda.
  - Índices dispersos: se supone que el índice almacena una entrada por cada bloque. Si el sistema crea un bloque nuevo, inserta el primer valor de la clave de búsqueda (en el orden de la clave de búsqueda) que aparezca en el nuevo bloque del índice. Por otra parte, si el nuevo registro tiene el menor valor de la clave de búsqueda en su bloque, el sistema actualiza la entrada del índice que apunta al bloque; si no, el sistema no realiza ningún cambio sobre el índice.
- **Borrado.** Para borrar un registro, primero se busca el índice a borrar. Las acciones que emprende el sistema a continuación dependen de si el índice es denso o disperso.
  - Índices densos:
    1. Si el registro borrado era el único registro con ese valor de la clave de búsqueda, el sistema borra el registro índice correspondiente del índice.
    2. En caso contrario, se emprenden las siguientes acciones:
       a. Si el registro índice almacena punteros a todos los registros con el mismo valor de la clave de búsqueda, el sistema borra del registro índice el puntero al registro borrado.
       b. En caso contrario, el registro índice almacena un puntero solo al primer registro con el valor de la clave de búsqueda. En este caso, si el registro borrado era el primer registro con el valor de la clave de búsqueda, el sistema actualiza el registro índice para apuntar al siguiente registro.
  - Índices dispersos:
    1. Si el índice no contiene un registro índice con el valor de la clave de búsqueda del registro borrado, no hay que hacer nada.
    2. En caso contrario, se emprenden las siguientes acciones:
       a. Si el registro borrado era el único registro con la clave de búsqueda, el sistema reemplaza el registro índice correspondiente con un registro índice para el siguiente valor de la clave de búsqueda (en el orden de la clave de búsqueda). Si el siguiente valor de la clave de búsqueda ya tiene una entrada en el índice, se borra en lugar de reemplazarlo.
       b. En caso contrario, si el registro índice para el valor de la clave de búsqueda apunta al registro a borrar, el sistema actualiza el registro índice para que apunte al siguiente registro con el mismo valor de la clave de búsqueda.

Los algoritmos de inserción y borrado para los índices multinivel se extienden de manera sencilla a partir del esquema descrito anteriormente. Al borrar o al insertar se actualiza el índice de nivel más bajo como se describió anteriormente. Por lo que respecta al índice del segundo nivel, el índice de nivel más bajo es simplemente un archivo de registros; así, si hay algún cambio en el índice de nivel más bajo, se tendrá que actualizar el índice del segundo nivel como ya se describió. La misma técnica se aplica al resto de niveles del índice, si los hubiera.

11.2.4. Índices secundarios

Los índices secundarios deben ser densos, con una entrada en el índice por cada valor de la clave de búsqueda, y un puntero a cada registro del archivo. Un índice con agrupación puede ser disperso, almacenando solo algunos de los valores de la clave de búsqueda, ya que siempre es posible encontrar registros con valores de la clave de búsqueda intermedios mediante un acceso secuencial a parte del archivo, como se describió antes. Si un índice secundario almacena solo algunos de los valores de la clave de búsqueda, los registros con los valores de la clave de búsqueda intermedios pueden estar en cualquier lugar del archivo y, en general, no se pueden encontrar sin explorar el archivo completo.

1 En los tiempos de índices basados en discos, cada nivel del índice se correspondía con una unidad de almacenamiento físico. Por tanto, tendríamos índices en los niveles de pista, cilindro y disco. Esta jerarquía no tiene sentido en la actualidad, ya que los subsistemas de discos ocultan los detalles físicos del almacenamiento en disco, y el número de discos y platos por disco es muy pequeño en comparación con el número de cilindros o bytes por pista.

Un índice secundario sobre una clave candidata es como un índice denso con agrupación, excepto en que los registros apuntados por los sucesivos valores del índice no están almacenados secuencialmente. Generalmente los índices secundarios están estructurados de manera diferente a como lo están los índices con agrupación. Si la clave de búsqueda de un índice con agrupación no es una clave candidata, es suficiente si el valor de cada entrada en el índice apunta al primer registro con ese valor en la clave de búsqueda, ya que los otros registros podrían ser alcanzados por una búsqueda secuencial del archivo.

En cambio, si la clave de búsqueda de un índice secundario no es una clave candidata, no sería suficiente apuntar solo al primer registro de cada valor de la clave. El resto de registros con el mismo valor de la clave de búsqueda podría estar en cualquier otro sitio del archivo, ya que los registros están ordenados según la clave de búsqueda del índice con agrupación, en vez de según la clave de búsqueda del índice secundario. Por tanto, un índice secundario debe contener punteros a todos los registros.

Se puede usar un nivel adicional de indirección para implementar los índices secundarios sobre claves de búsqueda que no sean claves candidatas. Los punteros en estos índices secundarios no apuntan directamente al archivo. En vez de eso, cada puntero apunta a un cajón que contiene punteros al archivo. En la Figura 11.6 se muestra la estructura de un índice secundario que usa un nivel adicional de indirección sobre el archivo *profesor*, teniendo como clave de búsqueda el *sueldo*.

Una búsqueda secuencial siguiendo el orden del índice de agrupamiento es eficiente porque los registros del archivo están guardados físicamente de la misma manera que está ordenado el índice. Sin embargo, no se puede (salvo en casos excepcionales) almacenar el archivo ordenado físicamente por el orden de la clave de búsqueda del índice con agrupación y la clave de búsqueda del índice secundario. Como el orden de la clave secundaria y el orden físico difieren, si se intenta examinar el archivo secuencialmente por el orden de la clave secundaria, es muy probable que la lectura de cada bloque suponga la lectura de un nuevo bloque del disco, lo cual es muy lento.

El procedimiento ya descrito para borrar e insertar se puede aplicar también a los índices secundarios; las acciones a emprender son las descritas para los índices densos que almacenan un puntero a cada registro del archivo. Si un archivo tiene varios índices, siempre que se modifique el archivo, se debe actualizar *cada* uno de ellos.

Los índices secundarios mejoran el rendimiento de las consultas que usan claves que no son la de búsqueda del índice con agrupación. Sin embargo, implican un tiempo adicional importante al modificar la base de datos. El diseñador de la base de datos debe decidir qué índices secundarios son deseables, según una estimación sobre la frecuencia relativa de consultas y modificaciones.

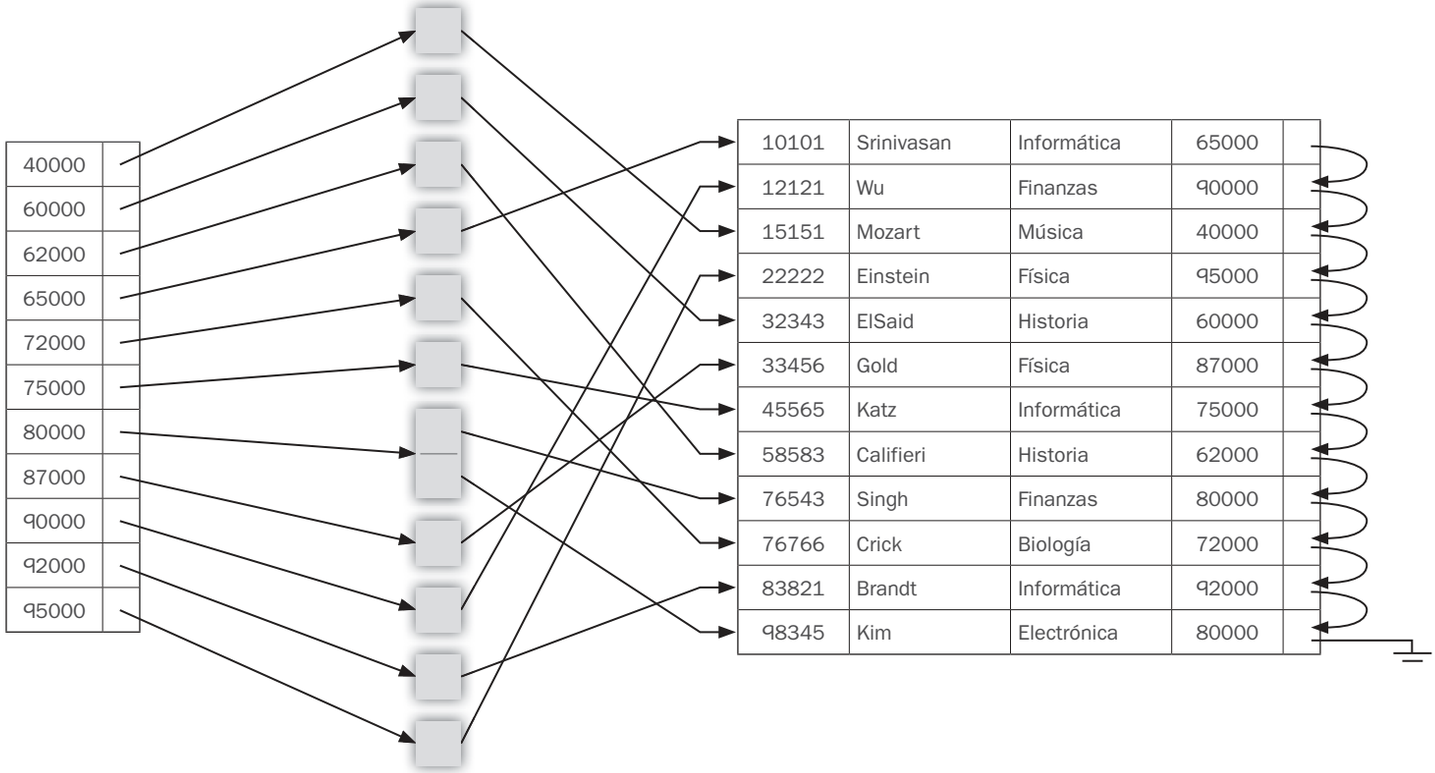


Figura 11.6. Índice secundario del archivo *profesor*, sobre la clave candidata *sueldo*.

CREACIÓN AUTOMÁTICA DE ÍNDICES

Si se declara que una relación tiene una clave primaria, la mayoría de las implementaciones de bases de datos crean automáticamente un índice de la clave primaria. Siempre que se inserte una tupla en la relación, se puede usar el índice para comprobar que no se infringe la restricción de clave primaria (es decir, no hay duplicados de la clave primaria). Sin el índice de la clave primaria, siempre que se insertara una tupla habría que leer toda la relación para asegurarse de que la restricción de clave primaria se satisface.

11.2.5. Índices con múltiples claves

Aunque los ejemplos vistos hasta ahora tengan solo un atributo en la clave de búsqueda, en general una clave de búsqueda puede tener más de un atributo. Cuando tiene más de un atributo se denomina **clave de búsqueda compuesta.** La estructura del índice es la misma que para el resto de otros índices; la única diferencia es que la clave de búsqueda no es un único atributo, sino una lista de ellos. La clave de búsqueda se puede representar como una tupla de valores, de la forma $(a_1 \ldots, a_n)$, donde los atributos indexados son $A_1 \ldots, A_n$. El orden de los valores de la clave de búsqueda es el *orden lexicográfico*. Por ejemplo, en el caso de dos atributos de clave de búsqueda, $(a_1, a_2) < (b_1, b_2)$ si $a_1 < b_1$ o $a_1 = b_1$ y $a_2 < b_2$. El orden lexicográfico es básicamente el mismo que el orden alfabético en el caso de palabras.

Como ejemplo, considere un índice con la relación *matricula*, con la clave de búsqueda (*asignatura_id*, *semestre*, *año*). Este índice sería útil para encontrar a todos los estudiantes que se hayan registrado para una determinada asignatura en un cierto semestre/año. Un índice ordenado con esta clave compuesta también se puede utilizar para responder a otros tipos de consultas de forma eficiente, como se verá en la Sección 11.5.2.

11.3. Archivos de índices de árboles B+

El inconveniente principal de la organización de un archivo secuencial indexado reside en que el rendimiento, tanto para buscar en el índice como para buscar secuencialmente a través de los datos, se degrada según crece el archivo. Aunque esta degradación se puede remediar reorganizando el archivo, no es deseable hacerlo con frecuencia.

La estructura de índice de **árbol B+** es la más extendida de las estructuras de índices que mantienen su eficiencia a pesar de la inserción y borrado de datos. Un índice de árbol B+ toma la forma de un **árbol equilibrado** en el que los caminos de la raíz a cada hoja del árbol son de la misma longitud. Cada nodo interno hoja tiene entre $\lceil n/2 \rceil$ y n hijos, siendo n fijo para cada árbol concreto.

Se verá que la estructura de árbol B+ implica una degradación del rendimiento al insertar y al borrar, además de un espacio extra. Esta sobrecarga es aceptable incluso en archivos con altas frecuencias de modificación, ya que se evita el coste de reorganizar el archivo. Además, puesto que los nodos podrían estar a lo sumo medio llenos (si tienen el mínimo número de hijos), se desperdicia algo de espacio. Este gasto de espacio adicional también es aceptable dados los beneficios en el rendimiento aportados por la estructura de árbol B+.

11.3.1. Estructura de un árbol B+

Un índice de árbol B+ es un índice multinivel pero con una estructura que difiere del índice multinivel de un archivo secuencial. En la Figura 11.7 se muestra un nodo típico de un árbol B+. Puede contener hasta $n - 1$ claves de búsqueda $K_1, K_2 \ldots, K_{n-1}$ y n punteros $P_1, P_2, \ldots, P_n$. Los valores de la clave de búsqueda de un nodo se mantienen ordenados; de forma que si $i < j$, entonces $K_i < K_j$.

Considere primero la estructura de los **nodos hoja.** Para $i = 1, 2, \ldots, n - 1$, el puntero P_i apunta a un registro del archivo con valor de la clave de búsqueda K_i. El puntero P_n tiene un propósito especial que se verá en breve.

En la Figura 11.8 se muestra un nodo hoja de un árbol B+ para el archivo *profesor*, en el que se ha elegido n igual a 4, y la clave de búsqueda es el *nombre*.

Ahora que se ha visto la estructura de un nodo hoja, se mostrará cómo los valores de la clave de búsqueda se asignan a nodos concretos. Cada hoja puede guardar hasta $n - 1$ valores. Está permitido que los nodos hoja contengan al menos $\lceil (n - 1)/2 \rceil$ valores. Con $n = 4$ en nuestro ejemplo, cada hoja debe contener al menos 2 valores, y como mucho 3 valores.

Los rangos de los valores en cada hoja no se solapan, excepto si hay valores de claves de búsqueda duplicados, en cuyo caso el valor puede estar presente en más de una hoja. Concretamente, si L_i y L_j son nodos hoja e $i < j$, entonces cada valor de la clave de búsqueda en L_i es menor o igual que cada valor de la clave de búsqueda en L_j. Si el índice de árbol B+ es un índice denso (como es habitual), cada valor de la clave de búsqueda debe aparecer en algún nodo hoja.

Ahora se puede explicar el uso del puntero P_n. Dado que existe un orden lineal en las hojas basado en los valores de la clave de búsqueda que contienen, se usa P_n para encadenar juntos los nodos hoja en el orden de la clave de búsqueda. Esta ordenación permite un procesamiento secuencial eficiente del archivo.

P_1	K_1	P_2	...	P_{n-1}	K_{n-1}	P_n

Figura 11.7. Nodo típico de un árbol B+.

Los **nodos internos** del árbol B+ forman un índice multinivel (disperso) sobre los nodos hoja. La estructura de los nodos internos es la misma que la de los nodos hoja, excepto que todos los punteros son punteros a nodos del árbol. Un nodo interno podría guardar hasta n punteros y *debe* guardar al menos $\lceil n/2 \rceil$ punteros. El número de punteros de un nodo se llama *grado de salida* del nodo.

Considere un nodo que contiene m punteros ($m \leq n$). Para $i = 2, 3, \ldots, m - 1$, el puntero P_i apunta al subárbol que contiene los valores de la clave de búsqueda menores que K_i y mayores o iguales que K_{i-1}. El puntero P_m apunta a la parte del subárbol que contiene los valores de la clave mayores o iguales que K_{m-1}, y el puntero P_1 apunta a la parte del subárbol que contiene los valores de la clave menores que K_1.

A diferencia de otros nodos internos, el nodo raíz puede tener menos de $\lceil n/2 \rceil$ punteros; sin embargo, debe tener al menos dos punteros, salvo que el árbol tenga un solo nodo. Siempre es posible construir un árbol B+, para cualquier n, que satisfaga los requisitos anteriores.

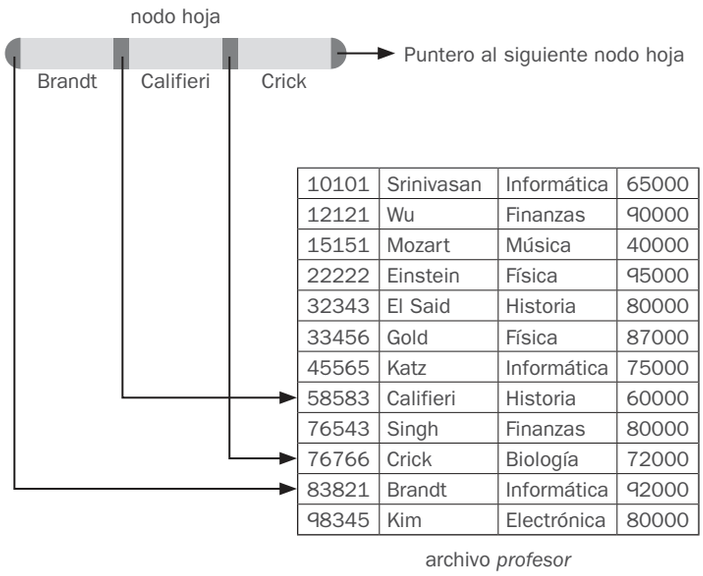


Figura 11.8. Nodo hoja para el índice del árbol B+ de *profesor* ($n = 4$).

En la Figura 11.9 se muestra un árbol B⁺ completo para el archivo *profesor* (con $n = 4$). Por simplicidad, se han abreviado los nombres de los profesores y se omiten los punteros al propio archivo y los punteros nulos; cualquier campo puntero de la figura que no tenga una flecha se supone que tiene un valor nulo.

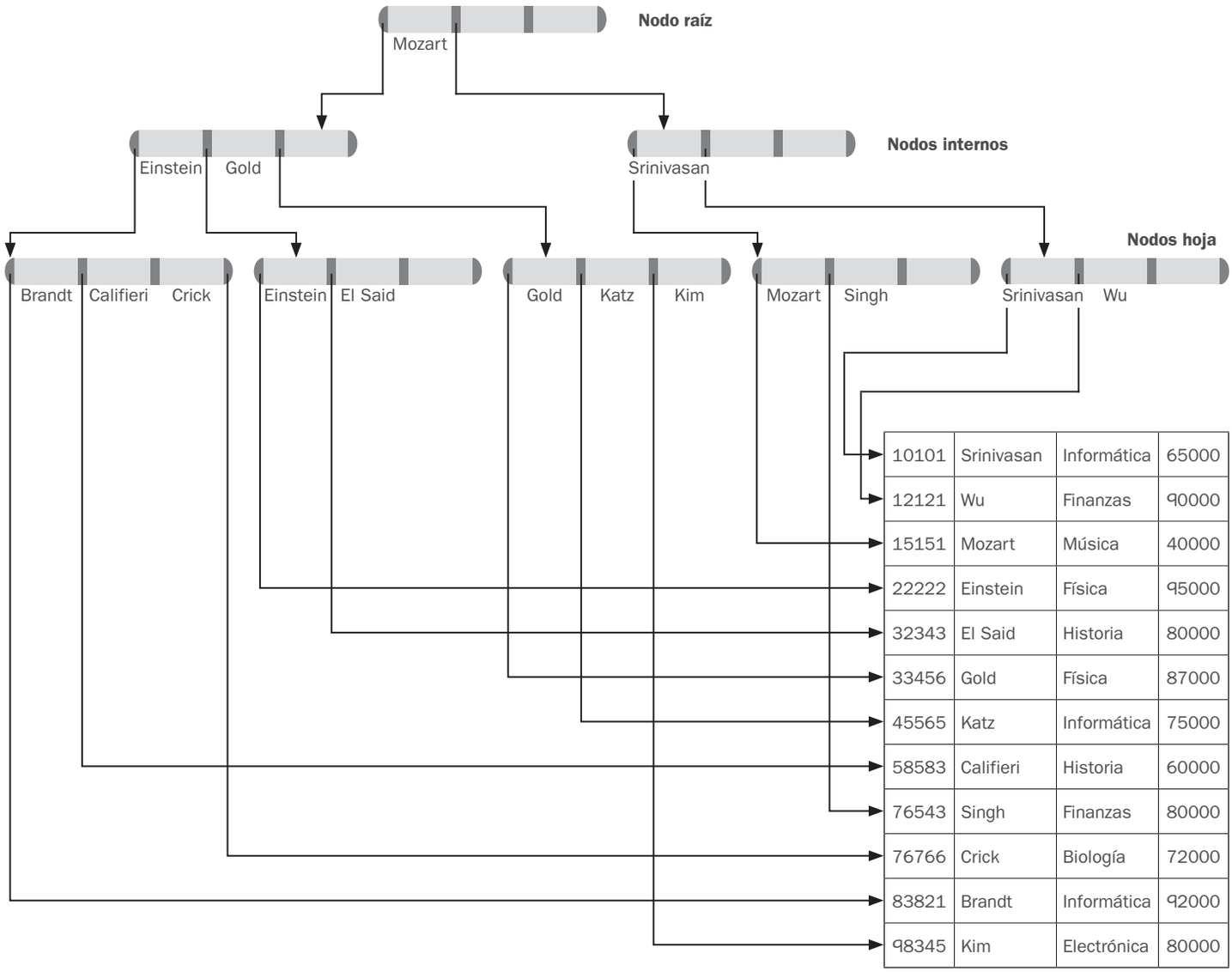


Figura 11.9. Árbol B⁺ para el archivo *profesor* ($n = 4$).

La Figura 11.10 muestra otro árbol B⁺ del archivo *profesor*, esta vez para $n = 6$. Al igual que antes, se han abreviado los nombres de los profesores para simplificar la presentación. Observe que la altura de este árbol es menor que la del anterior, que era de $n = 4$.

En todos los ejemplos mostrados de árboles B⁺, estos se encuentran equilibrados. Es decir, la longitud de cada camino desde la raíz a cada nodo hoja es la misma. Esta propiedad es un requisito de los árboles B⁺. De hecho, la «B» de árbol B⁺ proviene del inglés «*balanced*» (equilibrado).

Es esta propiedad de equilibrio de los árboles B⁺ la que asegura un buen rendimiento para las búsquedas, inserciones y borrados.

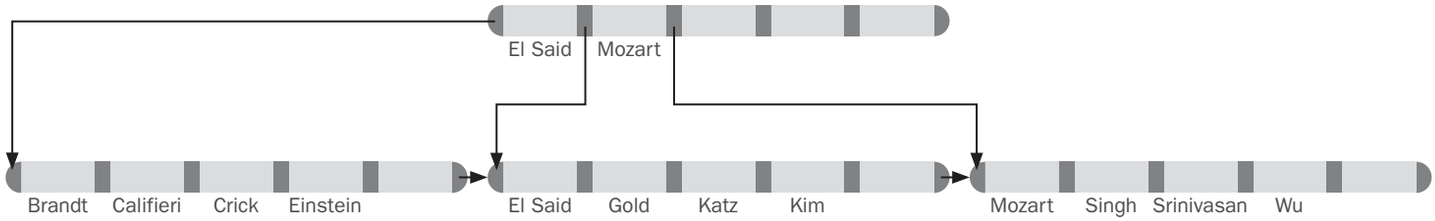


Figura 11.10. Árbol B⁺ para el archivo *profesor* con $n = 6$.

11.3.2. Consultas en árboles B+

Considere ahora cómo procesar consultas usando árboles B+. Suponga que se desean encontrar todos los registros cuyo valor de la clave de búsqueda sea V. La Figura 11.11 muestra el pseudocódigo de una función buscar() para hacerlo.

Intuitivamente, la función comienza con el nodo raíz del árbol y atraviesa este hacia abajo hasta que alcanza un nodo hoja que contendría el valor especificado si existe en el árbol. Concretamente, empezando con el nodo raíz como nodo actual, la función repite los siguientes pasos hasta que llega a un nodo raíz. Primero se examina el nodo actual buscando el menor i tal que el valor de la clave de búsqueda K_i sea mayor o igual que V. Suponga que se encuentra el valor; entonces, si K_i es igual a V el nodo actual se dispone como nodo al que apunta P_{i+1}; en otro caso $K_i > V$, y el nodo actual se dispone como nodo al que apunta P_i. Si no se encuentra el valor K_i, entonces claramente $V > K_{m-1}$, donde P_m es el último puntero no nulo en el nodo. En este caso el nodo actual se dispone como nodo al que apunta P_m. Este procedimiento se repite, recorriendo el árbol hacia abajo hasta alcanzar un nodo hoja.

```
function buscar(valor V)
/* Devuelve el nodo hoja C y el índice i tales que C.P_i apunta al primer registro
* con valor de clave de búsqueda igual a V */
  C = nodo raíz
  while (C no es un nodo hoja) begin
    i = menor número tal que V ≤ C.K_i
    if no existe tal número i then begin
      P_m = último puntero no nulo en el nodo
      C = C.P_m
    end
    else if (V = C.K_i)
      then C = C. P_i+1
    else C = C. P_i /* V < C.K_i */
  end
  /* C es un nodo hoja */
  Sea i el menor valor tal que K_i = V
  if existe ese valor i
    then return (C, i)
    else return null; /* No existe registro con el valor V */

procedure imprimirTodo(valor V)
/* escribe todos los registros con valor de clave de búsqueda igual a V */
  done = false;
  (L, i) = encontrar(V);
  if ((L, i) es null) return
  repeat
    repeat
      imprimirTodo el registro apuntado por L.P_i
      i = i + 1
    until (i > número de claves en L o L.K_i > V)
    if (i > número de claves en L)
      then L = L.P_n
      else done = true;
  until (done o L es null)
```

Figura 11.11. Consulta de un árbol B+.

En el nodo hoja, si existe un valor para la clave de búsqueda igual a V, sea K_i el primero de dichos valores; el puntero P_i dirige al registro con valor de la clave de búsqueda K_i. La función devuelve entonces el nodo L y el índice i. Si no se encuentra una clave de búsqueda con el valor V en el nodo hoja, no existe ningún registro con el valor de clave V en la relación y la función buscar() devuelve *null*, para indicar el resultado fallido.

Si existe al menos un registro con un valor de clave de búsqueda V (por ejemplo, si el índice no es una clave primaria), el procedimiento que llama a la función buscar() simplemente utiliza el puntero $L.P_i$ para obtener el registro y listo. Sin embargo, en el caso de que haya más de un registro que case, también hay que obtener el resto.

El procedimiento imprimirTodo que se muestra en la Figura 11.11 muestra cómo encontrar todos los registros con una determinada clave de búsqueda V. El procedimiento, en primer lugar pasa por el resto de las claves en el nodo L, para encontrar otros registros con el valor de la clave de búsqueda. Si el nodo L contiene al menos un valor de clave de búsqueda mayor que V, entonces no existen registros que coincidan con V. En otro caso, el siguiente registro hoja, al que apunta P_n, puede contener más entradas de V. Hay que continuar buscando a partir del nodo al que apunta P_n para encontrar más registros con el valor de clave de búsqueda V. Si el mayor de los valores de clave de búsqueda en el nodo al que apunta P_n también es V, habrá que revisar más hojas para encontrar todos los registros que casen. El bucle **repeat** de imprimirTodo es el que lleva a cabo la tarea de recorrer todos los nodos hoja hasta que se encuentren todos los registros que coincidan.

Una implementación real de buscar() tendría una versión con una interfaz iteradota similar a la que proporciona ResultSet de JDBC, que se vio en la Sección 5.1.1. Esta interfaz iteradora proporciona un método next(), que se llama de forma repetida para obtener los sucesivos registros con la clave de búsqueda indicada. El método next() recorrería las entradas en el nivel de las hojas, de forma similar a imprimirTodo, pero con cada llamada se avanza un solo paso y registra dónde va parando de forma que las llamadas sucesivas a next() recorran los sucesivos registros. Se omiten los detalles por simplicidad y se deja el pseudocódigo de la interfaz iteradora como ejercicio para el lector interesado.

Los árboles B+ también se pueden utilizar para encontrar todos los registros con valores de clave de búsqueda en un determinado intervalo (L, U). Por ejemplo, con un árbol B+ para el atributo *sueldo* de *profesor*, se podrían encontrar todos los registros de *profesor* con un sueldo en un intervalo como (50000, 100000) (en otras palabras, todos los sueldos entre 50000 y 100000). Estas consultas se llaman **consultas de intervalo.** Para ejecutar estas consultas se puede crear un procedimiento imprimirIntervalo(L, U), cuyo cuerpo es el mismo que el de imprimirTodo, excepto por lo siguiente: imprimirIntervalo llama a encontrar(L), en lugar de a encontrar(V), y después recorre los registros como en el procedimiento imprimirTodo, pero con la condición de parada $L.K_i > U$, en lugar de $L.K_i > V$.

Durante el procesamiento de una consulta se atraviesa el árbol desde la raíz hasta algún nodo hoja. Si existen N registros en el archivo, la longitud del camino recorrido no es mayor que $\lceil \log_{\lceil n/2 \rceil}(N) \rceil$.

En la práctica, solo hay que acceder a unos pocos nodos. Normalmente, si un nodo tiene el mismo tamaño que un bloque de disco, suele ser de 4 kilobytes. Con una clave de búsqueda de 12 bytes, y un puntero a disco de 8 bytes, n es de unos 200. Incluso con una estimación más conservadora de 32 bytes para el tamaño de la clave de búsqueda, n es de unos 100. Con $n = 100$, si existen 1.000.000 de valores de claves de búsqueda en el archivo, una búsqueda requiere acceder a solo $\lceil \log_{50}(1.000.000) \rceil = 4$ nodos. Por tanto, hay que leer, como mucho, cuatro nodos del disco para realizar la búsqueda. El nodo raíz es el nodo al que más se accede y es probable que se encuentre en la memoria intermedia, por lo que habitualmente solo hay que leer tres o menos nodos del disco.

Una diferencia importante entre las estructuras de árbol B+ y los árboles en memoria, tales como los árboles binarios, está en el tamaño de los nodos y, por tanto, la altura del árbol. En los árboles binarios, los nodos son pequeños y tienen, como mucho, dos punteros. En los árboles B+, los nodos son grandes —normalmente un bloque del disco— y pueden tener un gran número de punteros. Así, los árboles B+ tienden a ser bajos y anchos, en lugar de altos y estrechos como los árboles binarios. En un árbol binario equilibrado, el camino de una búsqueda puede tener una longitud de $\lceil \log_2(N) \rceil$, donde N es el número de valores de la clave de búsqueda. Con $N =$ 1.000.000, como en el ejemplo anterior, un árbol binario equilibrado necesita alrededor de 20 accesos a nodos. Si cada nodo estuviera en un bloque del disco distinto, serían necesarias 20 lecturas de bloques para procesar la búsqueda, en contraste con las cuatro lecturas del árbol B+. La diferencia es significativa, pues cada lectura de bloque requiere mover el brazo del disco y una lectura junto con el movimiento del brazo requiere unos 10 milisegundos en un disco típico.

11.3.3. Actualizaciones en árboles B+

Cuando un registro se inserta, o se borra, de una relación, hay que actualizar los índices de forma correspondiente. Recuerde que las actualizaciones de un registro se pueden modelar como un borrado del registro antiguo seguido de una inserción del registro actualizado. Por tanto, solo se va a considerar el caso de las inserciones y borrados.

El borrado y la inserción son operaciones más complicadas que las búsquedas, ya que podría ser necesario **dividir** un nodo que fuese demasiado grande como resultado de una inserción, o **fusionar** nodos si un nodo se volviera demasiado pequeño (menor que $\lceil n/2 \rceil$ punteros). Además, cuando se divide un nodo o se fusionan un par de ellos, se debe asegurar que el equilibrio del árbol se mantenga. Para presentar la idea que hay detrás del borrado y la inserción en un árbol B+, se asumirá que los nodos nunca serán demasiado grandes ni demasiado pequeños. Bajo esta suposición, el borrado y la inserción se realizan como se indica a continuación.

- **Inserción.** Usando la misma técnica que para buscar con la función encontrar() (Figura 11.11), se busca un nodo hoja en el que tendría que aparecer el valor de la clave de búsqueda. Entonces se inserta una entrada (es decir, el par valor de la clave de búsqueda y puntero al registro) en el nodo hoja, de tal manera que las claves de búsqueda permanezcan ordenadas.
- **Borrado.** Usando la misma técnica que para buscar, se busca el nodo hoja que contiene la entrada a borrar, realizando una búsqueda sobre el valor de la clave de búsqueda del registro borrado; si existen varias entradas con el mismo valor de clave de búsqueda, se busca entre todas las entradas con el mismo valor de la clave de búsqueda hasta que se encuentra la entrada que apunta al registro a borrar. Entonces se elimina la entrada desde el nodo hoja. Todas las entradas en el nodo hoja que están hacia la derecha de la entrada borrada se desplazan una posición a la izquierda, de forma que no dejan huecos en las entradas tras borrarla.

A continuación, se considera el caso general de la inserción y el borrado, cuando hay que realizar una división o fusión de nodos.

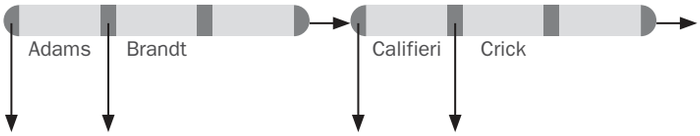


Figura 11.12. División de un nodo hoja tras la inserción de «Adams».

11.3.3.1. Inserción

A continuación se considera un ejemplo de inserción en el que se tiene que dividir un nodo. Suponga que se inserta un registro en la relación *profesor*, con el valor *nombre* «Adams». Entonces hay que insertar una entrada en el árbol B+ de la Figura 11.9, cuyo valor es «Adams». Usando el algoritmo de búsqueda, «Adams» debería aparecer en el nodo hoja que incluye «Brandt», «Califieri» y «Crick». No hay sitio en este nodo hoja para insertar el valor de la clave de búsqueda «Adams». Por tanto, se *divide* el nodo en otros dos nodos. En la Figura 11.12 se muestran los nodos hoja que resultan de insertar «Adams». Los valores de clave de búsqueda de «Adams» y «Brandt» están en una hoja y «Califieri» y «Crick» en la otra. En general, se toman los n valores de la clave de búsqueda (los $n - 1$ valores del nodo hoja más el valor que se inserta), y se dejan los $\lceil n/2 \rceil$ en el nodo existente y el resto de valores en el nuevo nodo.

Después de dividir el nodo hoja hay que insertar el nuevo nodo hoja en el árbol B+. En el ejemplo, el nuevo nodo «Califieri» es el valor más pequeño de la clave de búsqueda. Hay que insertar este valor de la clave de búsqueda, y un puntero al nuevo nodo, en el padre del nodo hoja dividido. En el árbol B+ de la Figura 11.13 se muestra el resultado de la inserción. Ha sido posible llevar a cabo esta inserción sin tener que dividir más, porque había sitio en el nodo padre para añadir una nueva entrada. En caso de no haber sitio, se dividiría el padre, teniendo que añadir una nueva entrada en el padre. En el peor de los casos, todos los nodos en el camino hacia la raíz se tendrían que dividir. Si la propia raíz se tuviera que dividir, el árbol hubiera sido más profundo.

La división de un nodo interno es un poco distinta a la de una hoja. En la Figura 11.14 se muestra el resultado de insertar un registro con la clave de búsqueda «Lamport» en el árbol que se muestra en la Figura 11.13. El nodo hoja en el que se va a insertar «Lamport» ya tiene las entradas «Gold», «Katz» y «Kim», y el nodo tiene que dividirse. El nuevo nodo de la derecha, resultado de la división, contiene los valores de las claves de búsqueda «Kim» y «Lamport». Hay que añadir una entrada (Kim, $n1$) al nodo padre, donde $n1$ es un puntero al nuevo nodo. Sin embargo, no existe espacio en el nodo padre para añadir una nueva entrada, y el nodo padre se tiene que dividir. Para ello, el nodo padre se expande conceptualmente de forma temporal, se añade la entrada y entonces se divide el nodo extendido.

Cuando un nodo interno saturado se divide, los punteros hijos se dividen entre el nodo original y el nuevo nodo creado; en el ejemplo, el nodo original es el de la izquierda con los primeros tres punteros, y el nuevo nodo creado es el de la derecha con los restantes dos punteros. Los valores de las claves de búsqueda se manejan, sin embargo, de forma algo diferente. Los valores de la clave de búsqueda están entre los punteros que se han movido al nodo de la derecha (en el ejemplo, el valor «Kim») junto con los punteros, mientras que los que permanecen en la izquierda (en el ejemplo, «Califieri» y «Einstein»), se mantienen sin cambios.

Sin embargo, el valor de la clave de búsqueda que está entre los punteros que permanecen en la izquierda y los punteros que se movieron a la derecha se trata de forma diferente. En el ejemplo, el valor de la clave de búsqueda «Gold» está entre los tres punteros que se encuentran en el nodo de la izquierda y los dos punteros que se movieron al nodo de la derecha. El valor «Gold» no se añade a ninguno de los nodos divididos. En lugar de ello, se añade una entrada (Gold, $n2$) al nodo padre, donde $n2$, es un puntero al nodo recién creado que es el resultado de la división. En este caso, el nodo padre es el nodo raíz, y tiene suficiente espacio para la nueva entrada.

La técnica general para la inserción en un árbol B+ es determinar el nodo hoja l en el que se debe producir la inserción. Si hay que dividirlo, se inserta el nuevo nodo en el padre del nodo l. Si esta inserción genera una división, se procede de forma recursiva hasta que una inserción no genere una división o se cree un nuevo nodo raíz.

En la Figura 11.16 se bosqueja el algoritmo de inserción en pseudocódigo. El procedimiento insertar() inserta un par valor de la clave y puntero en el índice usando los procedimientos auxiliares insertar_en_hoja e insertar_en_padre. En el pseudocódigo L, N, P y T denotan punteros a nodos y L se usa para denotar un nodo hoja. $L.K_i$ y $L.P_i$ denotan el i-ésimo valor y el i-ésimo puntero en el nodo L, respectivamente. $T.K_i$ y $T.P_i$ se usan de forma análoga. El pseudocódigo también emplea la función padre(N) para encontrar el padre del nodo N. Se puede obtener una lista de los nodos en el camino de la raíz a la hoja al buscar inicialmente el nodo hoja, y se puede usar después para encontrar eficazmente el padre de cualquier nodo del camino.

El procedimiento insertar_en_padre tiene como parámetros N, K' y N', donde el nodo N se ha dividido en N y N', siendo K' el valor mínimo en N'. El procedimiento modifica el padre de N para registrar la división.

Los procedimientos insertar_en_hoja e insertar_en_padre usan el área temporal de memoria T para almacenar el contenido del nodo que se está dividiendo. Los procedimientos se pueden modificar para que copien directamente los datos del nodo que se divide en el nodo que se crea, reduciendo el tiempo necesario para la copia. Sin embargo, el uso del espacio temporal T simplifica los procedimientos.

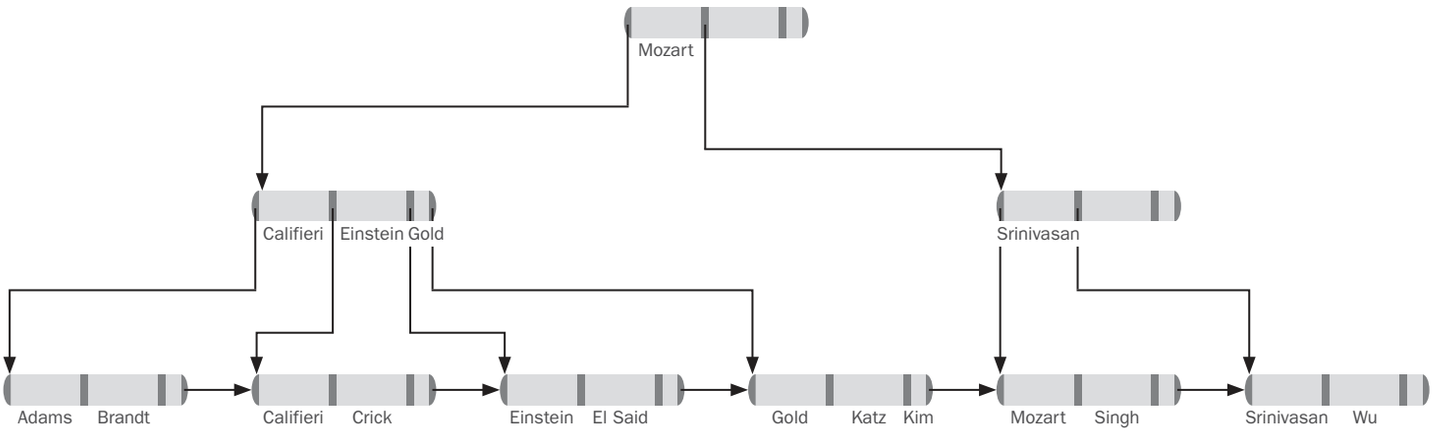


Figura 11.13. Inserción de «Adams» en el árbol B$^+$ de la Figura 11.9.

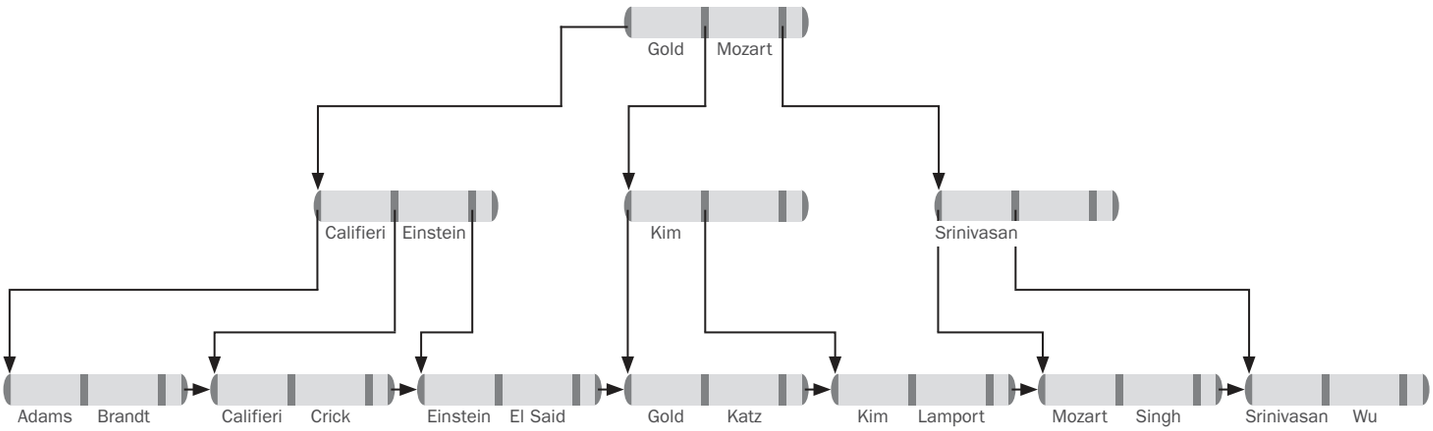


Figura 11.14. Inserción de «Lamport» en el árbol B$^+$ de la Figura 11.13.

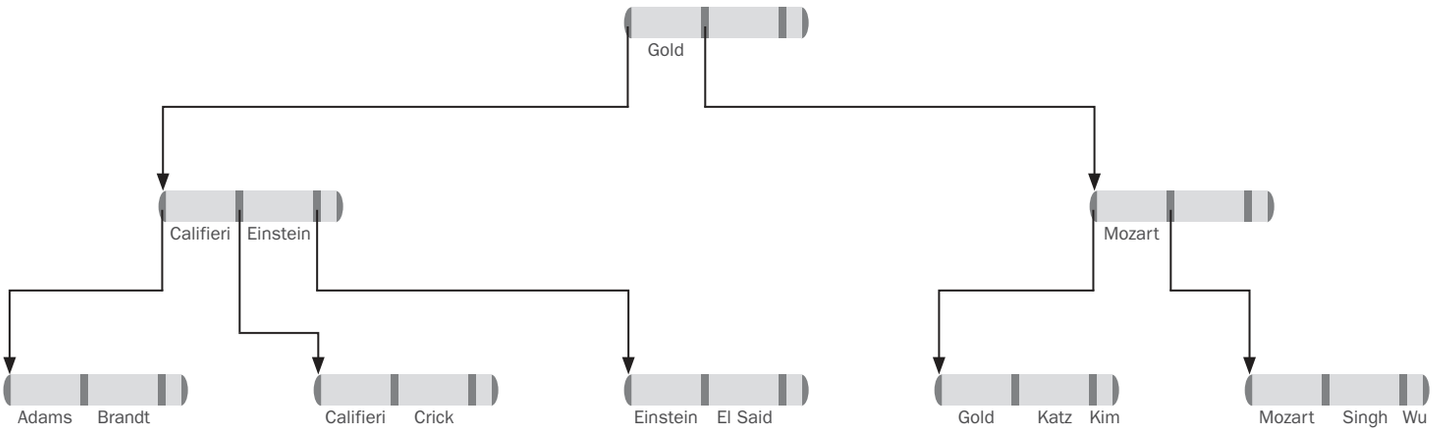


Figura 11.15. Borrado de «Srinivasan» del árbol B$^+$ de la Figura 11.13.

```
procedure insertar(valor K, puntero P)
  if (árbol está vacío) crea un nodo hoja vacío L, que es también la raíz
  else busca en nodo hoja L que debería contener el valor K
  if (L tiene menos de n − 1 valores de clave)
    then insertar_en_hoja (L, K, P)
    else begin /* L ya tiene n − 1 valores de la clave, dividirlo */
      Crear nodo L'
      Copiar L.P1 ... L.Kn−1 a un bloque de memoria T que pueda alma-
      cenar n pares (puntero, valor de la clave)
      insertar_en_hoja(T, K, P)
      L'.Pn = L.Pn; L.Pn = L'
      Borrar desde L.P1 hasta L.Kn−1 de L
      Copiar desde T.P1 hasta T.K⌈n/2⌉ de T en L empezando en L.P1
      Copiar T.P⌈n/2⌉+1 desde T.Kn de T en L' empezando en L'.P1
      Hacer que K' sea el valor mínimo de la clave en L'
      insertar_en_padre(L, K', L')
    end

procedure insertar_en_hoja(nodo L, valor K, puntero P)
  if (K < L.K1)
    then insertar P.K en L justo antes de L.P1
    else begin
      Sea Ki el mayor valor en L que es menor que K
      Insertar P.K en L justo tras T.Ki
    end

procedure insertar_en_padre(nodo N, valor K', nodo N')
  if (N es la raíz del árbol)
    then begin
      Crear un nuevo nodo R que contenga N, K', N' /* N and N' son
      punteros */
      Hacer R nodo raíz del árbol
      return
    end
  P = padre(N)
  if (P tiene menos de n punteros)
    then insertar(K', N') en P justo tras N
    else begin /* Dividir P */
      Copiar P a un bloque de memoria T donde quepan P y (K', N')
      Insertar (K', N') en T justo tras N
      Borrar todas las entradas de P; Crear un nodo P'
      Copiar T.P1 ... T.P⌈n/2⌉ en P
      K'' = T.K⌈n/2⌉
      Copiar T.P⌈n/2⌉+1 ... T.Pn+1 en P'
      insertar_en_padre(P, K'', P')
    end
```

Figura 11.16. Inserción de una entrada en un árbol B+.

11.3.3.2. Borrado

A continuación se consideran los borrados que provocan que el árbol se quede con muy pocos punteros. Primero se borra «Srinivasan» del árbol B+ de la Figura 11.13. El resultado se muestra en la Figura 11.15. Así se realiza el proceso de borrado. Para ello se localiza la entrada «Srinivasan» usando el algoritmo de búsqueda. Cuando se borra la entrada «Srinivasan» de su nodo hoja, la hoja se queda con una sola entrada. «Wu». Como en el ejemplo $n = 4$, y $1 < \lceil (n-1)/2 \rceil$, se debe mezclar el nodo con un nodo hermano o redistribuir las entradas entre los nodos, para asegurar que todos los nodos están al menos medio llenos. En el ejemplo, el nodo con la entrada «Wu» se puede mezclar con su nodo hermano de la izquierda. La mezcla se hace moviendo las entradas de ambos nodos al hermano de la izquierda, y borrando el nodo vacío de la derecha. Una vez se borra el nodo también hay que borrar la entrada en el nodo padre que apunta al nodo que se acaba de borrar.

En el ejemplo, la entrada a borrar es (Srinivasan, $n3$), donde $n3$ es un puntero a la hoja que contiene «Srinivasan». (En este caso la entrada a borrar en el nodo interno tiene el mismo valor que la que se borra de la hoja; podría no ser el caso en la mayoría de los nodos borrados). Tras borrar la entrada anterior, el nodo padre, que tiene un valor de clave de búsqueda «Srinivasan» y dos punteros, ahora tiene un puntero (el de más a la izquierda del nodo) y ningún valor de clave de búsqueda. Como $1 < \lceil (n-1)/2 \rceil$, para $n = 4$, el nodo padre no está lleno. (Para valores grandes de n, un nodo que no quedara lleno podría contener algunos valores y punteros).

En este caso se mira al nodo hermano; en el ejemplo, el único hermano es el nodo interno que contiene las claves «Califieri», «Einstein» y «Gold». Si es posible se intenta fusionar el nodo con su hermano. En este caso la fusión no es posible, ya que el nodo y su hermano en conjunto tienen cinco punteros, más que el máximo de cuatro. La solución en este caso es **redistribuir** los punteros entre el nodo y su hermano, de forma que tengan al menos $\lceil n/2 \rceil = 2$ punteros hijo. Para ello, se mueve el puntero de más a la derecha desde el hermano de la izquierda (el que apunta al nodo hoja que contiene «Mozart») al nodo hermano que no está lleno de su derecha. Sin embargo, el nodo hermano de su derecha ahora tendría dos punteros, su puntero de la izquierda y el nuevo puntero movido, sin ningún valor separándolos. De hecho, el valor que los separa no se encuentra en ninguno de los nodos, pero está en su nodo padre, entre los punteros de su nodo padre y su hermano. En el ejemplo, el valor «Mozart» separa los dos punteros, y se encuentra en el hermano de la derecha tras la redistribución. La redistribución de los punteros también implica que el valor «Mozart» en el padre ya no separa correctamente los valores de las claves de búsqueda en los dos hermanos. De hecho, el valor que ahora separa correctamente los valores de las claves de búsqueda en los dos nodos hermanos es el valor «Gold», que era el hermano de la izquierda antes de la redistribución.

El resultado, como se puede ver en el árbol B+ de la Figura 11.15, tras la redistribución de los punteros entre los hermanos, es que el valor «Gold» se ha movido hacia el padre, mientras que el valor que antes se encontraba allí, «Mozart», se ha movido hacia abajo, al hermano de la derecha.

A continuación se borran los valores «Singh» y «Wu» de clave de búsqueda del árbol B+ de la Figura 11.15. El resultado se muestra en la Figura 11.17. El borrado del primero no hace que el nodo hoja quede medio lleno, pero el borrado del segundo sí. No es posible mezclar el nodo medio lleno con su hermano, por lo que se lleva a cabo una redistribución de los valores, moviendo el valor de la clave de búsqueda «Kim» al nodo que contiene «Mozart», cuyo resultado se puede ver en la Figura 11.17. El valor que separa a los dos hermanos se ha actualizado en el padre, pasando de «Mozart» a «Kim».

A continuación se borra «Gold» del árbol anterior, y el resultado se muestra en la Figura 11.18. Este borrado genera una hoja menos de medio llena, que ahora se puede mezclar con su hermano. Al borrar una entrada del nodo padre (el nodo interno que contiene «Kim») hace que el padre quede menos de medio lleno (queda con un único puntero). Esta vez el nodo padre se puede mezclar con su hermano. Esta mezcla hace que el valor de la clave de búsqueda «Gold» se mueva hacia abajo desde el nodo padre al nodo mezclado. Como resultado de esta mezcla, se borra una entrada del padre, que resulta ser el nodo raíz del árbol. Y como resultado de este borrado, la raíz queda con un único puntero hijo y ningún valor de clave de búsqueda, incumpliendo la condición de que la raíz tenga al menos dos hijos. Entonces, se borra el nodo raíz y su único hijo se convierte en la raíz, con lo que el árbol B^+ reduce su altura en 1.

Hay que indicar que, como resultado del borrado, el valor clave que queda en un nodo interno del árbol B^+ puede que no esté presente en ninguno de los nodos hoja del árbol. Por ejemplo, en la Figura 11.18, el valor «Gold» se ha borrado del nivel de las hojas, pero aún sigue presente en un nodo interno.

En general, para borrar un valor en un árbol B^+ se realiza una búsqueda según el valor y se borra. Si el nodo es demasiado pequeño, se borra desde su padre. Este borrado se realiza como una aplicación recursiva del algoritmo de borrado hasta que se alcanza la raíz, hasta que un nodo padre queda lleno de manera adecuada después de borrar, o hasta aplicar una redistribución.

En la Figura 11.19 se describe el pseudocódigo para el borrado en un árbol B^+. El procedimiento intercambiar_variables(N, N') simplemente cambia de lugar los valores (punteros) de las variables N y N'; este cambio no afecta al árbol en sí mismo. El pseudocódigo utiliza la condición «muy pocos valores/punteros». Para nodos internos, este criterio quiere decir: menos que $\lceil n/2 \rceil$ punteros; para nodos hoja, quiere decir: menos que $\lceil (n-1)/2 \rceil$ valores. El pseudocódigo realiza la redistribución tomando prestada una sola entrada desde un nodo adyacente. También se puede redistribuir mediante la distribución equitativa de entradas entre dos nodos. El pseudocódigo hace referencia al borrado de una entrada (K, P) desde un nodo. En el caso de los nodos hoja, el puntero a una entrada realmente precede al valor de la clave; así, el puntero P precede al valor de la clave K. Para nodos internos, P sigue al valor de la clave K.

11.3.4. Claves de búsqueda que no son únicas

En una relación puede haber más de un registro que contenga el mismo valor de clave de búsqueda (es decir, dos o más registros que tienen los mismos valores para los atributos indexados), lo que se denomina **claves de búsqueda que no son únicas.**

Un problema con las claves de búsqueda que no son únicas es la eficiencia del borrado. Suponga que hay un valor de clave de búsqueda que existe un gran número de veces y se desea borrar uno de los registros. El borrado debe buscar en las entradas, potencialmente en muchos nodos hoja, hasta que encuentre la entrada correspondiente al registro concreto que se desea borrar.

Una solución sencilla a este problema, que se utiliza en la mayoría de los sistemas de bases de datos, es hacer que las claves de búsqueda sean únicas creando claves de búsqueda compuestas que contengan la clave de búsqueda original y otro atributo, que en conjunto sea único para todos los registros. El atributo extra puede ser un identificador de registro, que es un puntero al registro, u otro atributo cuyo valor sea único entre todos los registros con el mismo valor de clave de búsqueda.

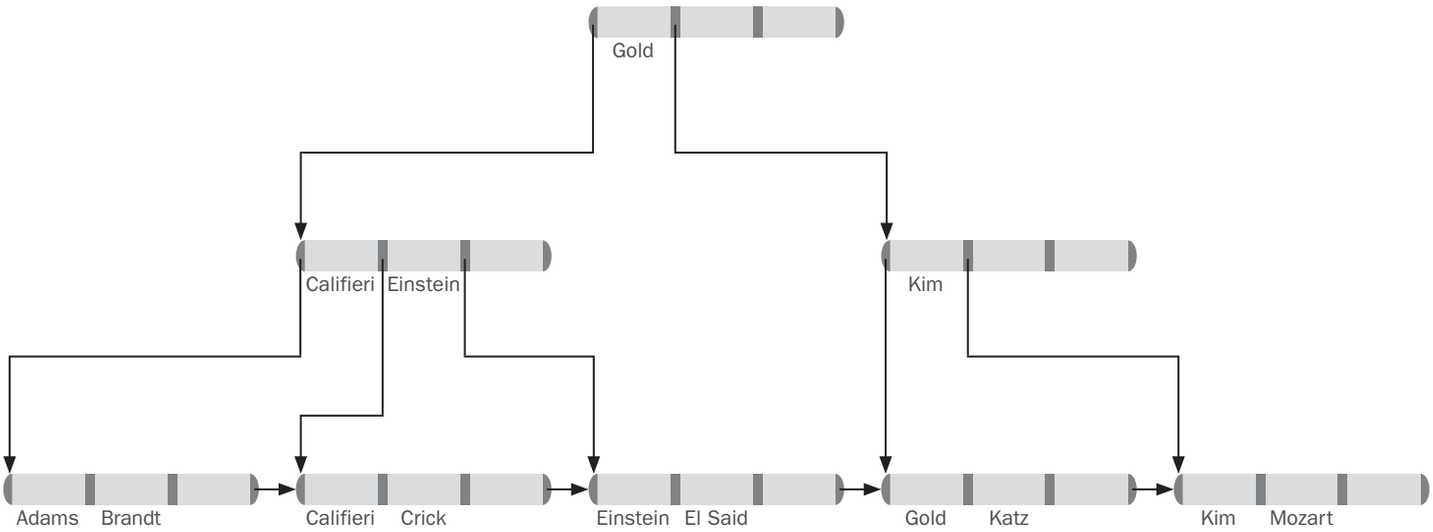


Figura 11.17. Borrado de«Singh» y «Wu» del árbol B^+ de la Figura 11.15.

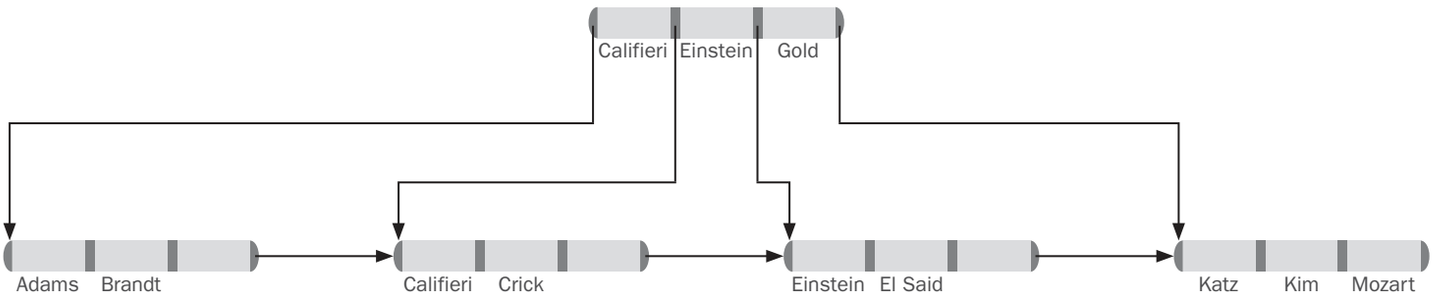


Figura 11.18. Borrado de «Gold» del árbol B^+ de la Figura 11.17.

```
procedure borrar(valor K, puntero P)
   buscar el nodo hoja L que contiene (K, P)
   borrar_entrada(L, K, P)

procedure borrar_entrada(nodo N, valor K, puntero P)
   borrar(K, P) de N
   if (N es la raíz and N solo le queda un hijo)
   then hacer el hijo de N la nueva raíz y borrar del árbol y borrar N
   else if (N tiene muy pocos valores/punteros) then begin
      Sea N' el anterior o siguiente hijo de padre(N)
      Sea K' el valor entre los punteros N y N' en padre(N)
      if (las entradas en N y N' caben en un único nodo)
         then begin /* Fusionar nodos */
            if (N es un predecesor de N') then intercambiar_variables(N, N')
            if (N no es una hoja)
               then añadir K' y todos los punteros y valores de N a N'
               else añadir todos los pares (K_i, P_i) en N a N';
                    hacer N'.P_n = N.P_n
            borrar_entrada(padre(N), K', N), borrar nodo N
         end
      else begin /* Redistribución: prestar una entrada desde N' */
         if (N' es un predecesor de N) then begin
            if (N es un nodo interno) then begin
               sea m tal que N'.P_m es el último puntero en N'
               borrar(N'.K_m-1, N'.P_m) de N'
               insertar(N'.P_m, K') como primer puntero y valor en N, desplazando otros punteros y valores a la derecha
               sustituir K' en padre(N) por N'.K_m-1
            end
            else begin
               sea m tal que (N'.P_m, N'.K_m) es el último par puntero/valor en N'
               borrar(N'.P_m, N'.K_m) de N'
               insertar(N'.P_m, N'.K_m) como primer puntero y valor en N, desplazando otros punteros y valores a la derecha
               sustituir K' en padre(N) por N'.K_m
            end
         end
         else... simétrico al caso then...
      end
   end
```

Figura 11.19. Borrado de una entrada de un árbol B+.

El atributo extra se denomina atributo «**que hace único**» (*uniquifier*). Cuando se borra un registro, el valor de la clave de búsqueda se calcula a partir del registro y se usa para actualizar el índice. Como el valor es único, se puede buscar el nivel de hoja correspondiente con un único recorrido desde la raíz hasta la hoja, sin acceder más veces al nivel hoja. De esta forma el borrado se hace eficientemente.

Una búsqueda con el atributo de clave de búsqueda original ignora el valor de atributo «que hace único» cuando se comparan los valores de las claves de búsqueda.

Con claves de búsqueda que no son únicas, la estructura de árbol B+ guarda cada valor de clave tantas veces como haya registros que contengan dicha clave. Una alternativa es guardar cada valor de clave solo una vez y mantener un cajón (o lista) de punteros a registros con ese valor de clave de búsqueda, para manejar las claves de búsqueda que no son únicas. Esta forma es más eficiente en espacio, ya que guarda el valor de la clave solo una vez; sin embargo, crea varias complicaciones cuando se implementan los árboles B+. Si los cajones se mantienen en los nodos hoja, hace falta código extra para tratar con los cajones de tamaño variable y para tratar con cajones que son mayores que el tamaño del nodo hoja. Si los cajones se guardan en bloques separados, hacen falta operaciones adicionales de E/S para cargar los registros. Además de estos problemas, los cajones también presentan ineficiencias en el borrado de registros si el valor de clave de búsqueda aparece un gran número de veces.

11.3.5. Complejidad de las actualizaciones de los árboles B+

Aunque las operaciones de inserción y borrado en los árboles B+ son complejas, requieren relativamente pocas operaciones de E/S, lo que supone una gran ventaja, ya que las operaciones de E/S son muy costosas. Se puede demostrar que el número de operaciones de E/S en el caso peor para una inserción es proporcional a $\log_{\lceil n/2 \rceil}(N)$, donde n es el número máximo de punteros en un nodo y N es el número de registros en el archivo que se indexan.

La complejidad del caso peor del procedimiento de borrado también es proporcional a $\log_{\lceil n/2 \rceil}(N)$, dado que no existen valores duplicados de las claves de búsqueda. Si existen valores duplicados, al borrar hay que buscar en múltiples registros con el mismo valor de la clave de búsqueda para encontrar la entrada correcta a borrar, lo que puede ser ineficiente. Sin embargo, construyendo una clave de búsqueda única y añadiendo un atributo que lo haga único, como se ha descrito en la Sección 11.3.4, se asegura que la complejidad del caso peor en el borrado sea igual si la clave de búsqueda original no es única.

En otras palabras, el coste de las operaciones de inserción y borrado en términos de operaciones de E/S es proporcional a la altura del árbol B+, y por tanto es baja. Es la rapidez de las operaciones sobre los árboles B+ lo que hace que se use habitualmente una estructura de índices en las implementaciones de las bases de datos.

En la práctica, las operaciones sobre los árboles B+ generan menos operaciones de E/S que en el caso peor. Con nodos con grado de salida de 100, y suponiendo que el acceso a los nodos hojas se distribuye uniformemente, es 100 veces más probable que se acceda al padre de un nodo hoja que al nodo hoja. De la misma forma, con el mismo grado de salida, el número total de nodos internos en el árbol B+ sería unos pocos más de 1/100 del número de nodos hoja. Por ello, con tamaños normales de memoria de varios gigabytes, para los árboles B+ que se usan frecuentemente, incluso si la relación es muy grande es muy probable que la mayoría de los nodos internos ya se encuentren en la memoria intermedia cuando se necesite acceder a los mismos. Por tanto, normalmente solo hace falta una o dos operaciones de E/S para realizar una búsqueda. Para las actualizaciones, la probabilidad de dividir un nodo es, por la misma razón, muy pequeña. Dependiendo del orden de las inserciones, con nodos con grado de salida de 100, solo en una de cada 100 a 1 de cada 50 inserciones habrá que dividir el nodo, siendo necesario escribir más de un bloque. El resultado es que, en promedio, una inserción requiere un poco más de una operación de E/S para escribir los bloques actualizados.

Aunque los árboles B+ garantizan que los nodos se encuentran al menos medio llenos, si las entradas se insertan de forma aleatoria se puede esperar que, en promedio, se encuentren llenos hasta más de las dos terceras partes. Si las entradas se insertan en orden, los nodos se encontrarán llenos solo hasta la mitad. (Se deja como ejercicio para el lector descubrir por qué los nodos quedarán solo medio llenos en este último caso).

11.4. Extensiones de los árboles B+

En esta sección se tratan diversas extensiones y variaciones de la estructura de índices de árboles B+.

11.4.1. Organización de archivos con árboles B+

Como se mencionó en la Sección 11.3, el mayor inconveniente de la organización de archivos secuenciales de índices es la degradación del rendimiento según crece el archivo: con el crecimiento, un porcentaje mayor de registros índice y registros reales se quedan fuera de orden y se almacenan en bloques de desbordamiento. La degradación de las búsquedas en el índice se resuelve mediante el uso de índices de árbol B+ en el archivo. También se soluciona el problema de la degradación al almacenar los registros reales utilizando el nivel de hoja del árbol B+ para almacenar los registros reales en los bloques. En estas estructuras, la estructura del árbol B+ se usa no solo como un índice, sino también como un organizador de los registros dentro del archivo. En la **organización de archivos con árboles B+**, los nodos hoja del árbol almacenan registros, en lugar de almacenar punteros a registros. En la Figura 11.20 se muestra un ejemplo de la organización de un archivo con un árbol B+. Ya que los registros son normalmente más grandes que los punteros, el número máximo de registros que se pueden almacenar en un nodo hoja es menor que el número de punteros en un nodo interno. Sin embargo, todavía se requiere que los nodos hoja estén al menos medio llenos.

La inserción y el borrado de registros de una organización de archivos con árboles B+ se manejan de la misma forma que la inserción y el borrado de entradas de índices de árboles B+. Cuando se inserta un registro con un valor de clave de búsqueda v, el sistema localiza el bloque que debería contener el registro buscando en el árbol B+ la mayor de las claves del árbol que sea $\leq v$. Si dicho bloque tiene espacio suficiente para el registro, el sistema guarda el registro en el bloque. Si no, como en la inserción en un árbol B+, el sistema divide el bloque en dos y redistribuye los registros entre ellos (en el orden de claves del árbol B+) para crear espacio para el nuevo registro. Esta división se propaga hacia arriba en el árbol B+ de la forma habitual. Cuando se borra un registro, el sistema primero lo elimina del bloque que lo contiene. Si el bloque B queda menos que medio lleno, los registros de B se redistribuyen con los registros de un bloque adyacente B'. Suponiendo registros de tamaño fijo, cada bloque contendrá al menos la mitad del número máximo de registros que pueda contener. El sistema actualiza los nodos internos del árbol B+ de la forma habitual.

Cuando se emplean árboles B+ para la organización de archivos, es especialmente importante la utilización del espacio, ya que el espacio que ocupan los registros probablemente sea mucho mayor que el espacio que ocupan las claves y punteros. Se puede mejorar la utilización del espacio de un árbol B+ implicando a más nodos hermanos durante los procesos de redistribución en las divisiones y mezclas. La técnica se puede usar tanto para los nodos hoja como para los nodos internos, y funciona como sigue:

Durante una inserción, si un nodo está completo el sistema intenta redistribuir algunas de sus entradas a uno de los nodos adyacentes, para conseguir espacio para la nueva entrada. Si este intento falla debido a que los nodos adyacentes también están llenos, el sistema divide el nodo y las entradas de forma equitativa entre uno de los nodos adyacentes y los dos nodos que se obtienen de la división del nodo original. Como los tres nodos juntos contienen un registro más que los que caben en dos nodos, cada nodo estará lleno dos tercios. De forma más precisa, cada nodo tendrá al menos $\lfloor 2n/3 \rfloor$ entradas donde n es el número máximo de entradas que puede contener un nodo ($\lfloor x \rfloor$ indica el mayor entero que es menor o igual a x; es decir se elimina la parte decimal, si existe).

Durante el borrado de un registro, si la ocupación de un nodo baja de $\lfloor 2n/3 \rfloor$, el sistema intenta tomar prestada una entrada de uno de sus nodos hermanos. Si ambos hermanos tienen $\lfloor 2n/3 \rfloor$ registros, en lugar de tomar prestada una entrada, el sistema redistribuye las entradas en el nodo y en los dos hermanos de forma equitativa entre los dos, y borra el tercer nodo. Se puede utilizar esta forma porque el número total de entradas es $3\lfloor 2n/3 \rfloor - 1$, que es menor que $2n$. Con tres nodos adyacentes para redistribuir las entradas, se puede garantizar que cada nodo tiene $\lfloor 3n/4 \rfloor$ entradas. En general, si m nodos ($m - 1$ hermanos) se utilizan para la redistribución, se garantiza que cada nodo contiene al menos $\lfloor (m-1)n/m \rfloor$ entradas. Sin embargo, el coste de actualización se va haciendo cada vez mayor según se utilizan más nodos hermanos en la redistribución.

Fíjese que en un índice con árbol B+ o en una organización con archivos, los nodos hoja que son adyacentes en el árbol pueden encontrarse en sitios diferentes del disco. Cuando se crea una nueva organización en disco con un conjunto de registros, se pueden asignar los bloques que son contiguos en el disco para que sean contiguos en el árbol. De esta forma una búsqueda secuencial de nodos hoja se correspondería con prácticamente una búsqueda secuencial en el disco. Según se vayan realizando inserciones y borrados en el árbol se irá perdiendo esta secuencialidad y, cada vez más, un acceso secuencial en el árbol tendrá que esperar por el tiempo de búsqueda en disco. Podría hacer falta reconstruir el índice para restaurar la secuencialidad.

Las organizaciones de archivos con árboles B+ también sirven para guardar objetos grandes, como los clobs y blobs de SQL que sean mayores que un bloque de disco, de hasta varios gigabytes. Objetos tan grandes se pueden guardar dividiéndolos en secuencias de registros más pequeños de acuerdo con una organización de archivos con árboles B+. Los registros se pueden numerar de forma secuencial o por el desplazamiento que tengan en el objeto, y el número de registro se puede utilizar como clave de búsqueda.

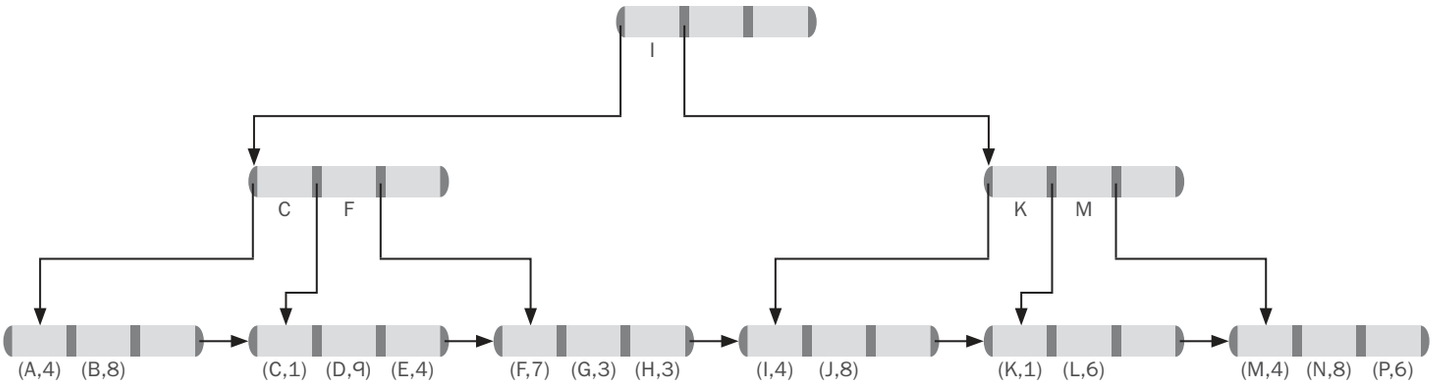


Figura 11.20. Organización de archivos con árboles B+.

11.4.2. Índices secundarios y reubicación de registros

Algunas organizaciones de archivos, como los árboles B^+, pueden cambiar la ubicación de los registros aunque estos no se hayan modificado. A modo de ejemplo, considérese que cuando en una organización de archivo de árbol B^+ se divide un nodo hoja, varios registros se trasladan a un nuevo nodo. En esos casos hay que actualizar todos los índices secundarios que almacenan punteros a los registros reubicados, aunque sus contenidos no hayan cambiado. Cada nodo hoja puede contener gran número de registros, y cada uno de ellos puede estar en diferentes ubicaciones de cada índice secundario. Así, la división de un nodo hoja puede exigir decenas o incluso centenas de operaciones de E/S para actualizar todos los índices secundarios afectados, lo que puede convertirla en una operación muy costosa.

A continuación se explica una técnica para resolver este problema. En los índices secundarios, en lugar de punteros a los registros indexados se almacenan los valores de los atributos de la clave de búsqueda del índice primario. Por ejemplo, suponga que se tiene un índice primario sobre el atributo *ID* de la relación *profesor*; un índice secundario sobre *nombre_dept* almacenaría con cada nombre de departamento una lista de valores de *ID* de profesores de los registros correspondientes, en lugar de almacenar punteros a los registros.

Por tanto, la reubicación de los registros debida a la división de los nodos hoja no necesita ninguna actualización de los índices secundarios. Sin embargo, localizar un registro mediante el índice secundario exige ahora dos pasos: primero se utiliza el índice secundario para buscar los valores de la clave de búsqueda del índice primario y luego se utiliza el índice primario para buscar los registros correspondientes.

Este enfoque reduce en gran medida el coste de actualización de los índices debido a la reorganización de los archivos, aunque incrementa el coste de acceso a los datos mediante un índice secundario.

11.4.3. Índices sobre cadenas de caracteres

La creación de índices de árboles B^+ sobre atributos de tipo cadena de caracteres plantea dos problemas. El primero es que las cadenas pueden ser de longitud variable. El segundo es que pueden ser largas, lo que produce un grado de salida bajo y a una altura del árbol incrementada de manera acorde.

Con las claves de búsqueda de longitud variable varios nodos pueden tener diferentes grados de salida, incluso estando llenos. Un nodo se debe dividir si está lleno, es decir, si no hay espacio para añadir una nueva entrada, independientemente de cuántas entradas contenga.

Análogamente, los nodos se pueden fusionar o las entradas se pueden redistribuir dependiendo de la fracción de espacio que se use en los nodos, en lugar de basarse en el número máximo de entradas que el nodo pueda contener.

Se puede incrementar el grado de salida usando la técnica denominada **compresión del prefijo.** Con esta compresión no se almacena la clave de búsqueda completa en los nodos internos, solo se almacena el prefijo de la clave de búsqueda que sea suficiente para distinguir las claves de los subárboles bajo ella. Por ejemplo, si se indexa el nombre, el valor de la clave en un nodo interno podría ser un prefijo del nombre; sería suficiente almacenar «Silb» en un nodo interno en lugar del nombre completo «Silberschatz» si los valores más próximos en los dos subárboles bajo esta clave son, por ejemplo, «Silas» y «Silver» respectivamente.

11.4.4. Carga en bruto de índices de árboles B^+

Como ya se ha tratado, la inserción de un registro en un árbol B^+ requiere de un cierto número de operaciones de E/S que en el caso peor es proporcional a la altura del árbol, que normalmente es pequeña (de cinco o menos, incluso para relaciones muy grandes).

Suponga un caso en el que se desea construir un árbol B^+ de una gran relación. Suponga que la relación es significativamente más grande que la memoria principal y que se desea construir un índice sin agrupamiento sobre la relación, de forma que el índice es también mayor que la memoria principal. En este caso, según se recorre la relación y se añaden entradas al árbol B^+, es muy probable que los nodos hoja a los que se accede no estén en la memoria intermedia de la base de datos cuando se accede a ellos, ya que no existe ningún orden particular en las entradas. Con estos accesos aleatorios a los bloques, cada vez que se añade una entrada a una hoja se requiere una búsqueda en disco para obtener el bloque que contiene el nodo hoja. El bloque probablemente sea expulsado de la memoria intermedia del disco antes de que se añada otra entrada al bloque, lo que conlleva otra búsqueda en disco para escribir de nuevo el bloque en el disco. Por tanto, se puede requerir una lectura y una escritura aleatorias para cada entrada que se inserte.

Por ejemplo, si la relación tiene 100 millones de registros y cada operación de E/S tarda unos 10 milisegundos, se tardaría al menos un millón de segundos en construir el índice, contando solo el coste de la lectura de los nodos hijo, sin contar el coste de escribir las actualizaciones de nuevo en el disco. Es un tiempo demasiado grande; en comparación, si cada registro ocupa 100 bytes y el subsistema de disco puede transferir datos a 50 megabytes por segundo, solo tardaría 200 segundos en leer toda la relación.

La inserción de un gran número de entradas a la vez en un índice se denomina **carga en bruto** del índice. Una forma eficiente de realizar una carga en bruto de un índice es la siguiente: en primer lugar se crea un archivo temporal que contiene las entradas del índice de la relación, después se ordena el archivo por la clave de búsqueda del índice que se va a construir y, finalmente, se recorre el archivo ordenado y se insertan las entradas en el índice. Existen algoritmos eficientes para ordenar grandes relaciones, que se describen más adelante en la Sección 12.4, que pueden ordenar incluso un archivo muy grande con un coste de E/S comparable al de leer el archivo unas pocas veces, suponiendo que se dispone de una cantidad razonable de memoria principal.

Supone una ventaja significativa ordenar las entradas antes de insertarlas en un árbol B^+. Cuando las entradas se insertan ordenadas, todas las entradas que van a un nodo hoja en concreto aparecen de forma consecutiva, por lo que el nodo hoja se escribe solo una vez; nunca hay que leer los nodos desde el disco durante una carga en bruto, si el árbol B^+ estaba vacío al empezar. Cada nodo hoja solo genera una operación de E/S incluso aunque se inserten muchas entradas en el nodo. Si cada nodo hoja contiene 100 entradas, el nivel de hoja contendrá un millón de nodos, lo que genera un millón de operaciones de E/S para crear el nivel de hoja. Incluso estas operaciones de E/S se espera que sean secuenciales si se van asignando nodos hoja sucesivos en bloques de disco sucesivos y, por tanto, se requerirán pocas búsquedas en disco. Con los discos actuales, 1 milisegundo por bloque es una estimación razonable para la mayoría de las operaciones de E/S secuenciales, en contraste con los 10 milisegundos por bloque de las operaciones de E/S aleatorias.

Más adelante se estudiará el coste de ordenar una relación grande, en la Sección 12.4, pero como estimación, el índice que habría tardado en construirse un millón de segundos, se construirá en menos de 1000 segundos ordenando las entradas antes de insertarlas en el árbol B^+, en contraste con más de un millón de segundos si se insertan de forma aleatoria.

Si el árbol B⁺ está inicialmente vacío, se puede construir más rápido haciéndolo de abajo arriba, desde el nivel de hojas, en lugar de utilizar el procedimiento habitual de inserción. En la **construcción del árbol B⁺ desde abajo,** tras ordenar las entradas como ya se ha descrito, se trocean las entradas ordenadas en bloques, manteniendo tantas entradas por bloque como quepan en un bloque; los bloques resultantes forman el nivel de hoja del árbol B⁺. El valor mínimo de cada bloque junto con el puntero al bloque se utiliza para crear las entradas del siguiente nivel del árbol B⁺, apuntando a los bloques hoja. Cada nivel adicional del árbol se construye de forma similar usando los valores mínimos asociados con cada nodo del nivel inferior, hasta que se crea la raíz. Se dejan los detalles como ejercicio para el lector.

La mayoría de los sistemas de bases de datos implementan técnicas eficientes basadas en la ordenación de las entradas y la construcción desde abajo cuando crean un índice para una relación, aunque usen el procedimiento de inserción normal cuando se añaden las tuplas una a una a una relación con un índice existente. Algunos sistemas de bases de datos recomiendan que si se van a añadir un gran número de tuplas a la vez a una relación existente, se deberían eliminar los índices de la relación (si son distintos del índice sobre la clave primaria), y volverlos a crear después de haber insertado las tuplas, para aprovechar las técnicas de carga en bruto eficientes.

11.4.5. Archivos de índices de árbol B

Los **índices de árbol B** son similares a los índices de árbol B⁺. La diferencia principal entre los dos enfoques es que un árbol B elimina el almacenamiento redundante de los valores de la clave de búsqueda. En el árbol B⁺ de la Figura 11.13, las claves de búsqueda «Califieri», «Einstein», «Gold», «Mozart» y «Srinivasan» aparecen en nodos internos, además de aparecer en nodo hoja. Cada valor de clave de búsqueda aparece en algún nodo hoja; algunos se repiten en nodos internos.

Los árboles B permiten que los valores de la clave de búsqueda aparezcan solamente una vez (si son únicos), al contrario que en el árbol B⁺, en el que un valor puede aparecer en un nodo interno, además de en un nodo hoja. En la Figura 11.21 se muestra un árbol B que representa las mismas claves de búsqueda que el árbol B⁺ de la Figura 11.13. Como las claves de búsqueda no están repetidas en el árbol B, sería posible almacenar el índice usando menos nodos del árbol que con el correspondiente índice de árbol B⁺. Sin embargo, puesto que las claves de búsqueda que aparecen en los nodos internos no aparecen en ninguna otra parte del árbol B, es necesario incluir un campo adicional para un puntero por cada clave de búsqueda de un nodo interno. Estos punteros adicionales apuntan a registros del archivo o a los cajones de la clave de búsqueda asociada.

Hay que destacar que muchos manuales de sistemas de bases de datos, artículos de la literatura y profesionales de la industria utilizan el término árbol B para referirse a la estructura de datos que llamamos árbol B⁺. De hecho, seria justo indicar que el término árbol B es sinónimo de árbol B⁺. Sin embargo, en este libro se usan los términos árbol B y árbol B⁺ como se definieron originalmente, para evitar confusiones entre las dos estructuras de datos.

En la Figura 11.22a aparece un nodo hoja generalizado de un árbol B; en la Figura 11.22b aparece un nodo interno. Los nodos hoja son como en los árboles B⁺. En los nodos internos los punteros P_i son los punteros del árbol que se utilizan también para los árboles B⁺, mientras que los punteros B_i en los nodos internos son punteros a cajones o registros del archivo. En la figura del árbol B generalizado hay $n - 1$ claves en el nodo hoja, mientras que hay $m - 1$ claves en el nodo interno. Esta discrepancia se produce porque los nodos internos deben incluir los punteros B_i, y de esta manera se reduce el número de claves de búsqueda que pueden contener estos nodos. Claramente, $m < n$, pero la relación exacta entre m y n depende del tamaño relativo de las claves de búsqueda y de los punteros.

El número de nodos a los que se accede en una búsqueda en un árbol B depende de dónde esté situada la clave de búsqueda. Una búsqueda en un árbol B⁺ requiere atravesar un camino desde la raíz del árbol hasta algún nodo hoja. En cambio, algunas veces es posible encontrar en un árbol B el valor deseado antes de alcanzar el nodo hoja. Sin embargo, hay que realizar aproximadamente n accesos según cuántas claves haya almacenadas tanto en el nivel de hoja de un árbol B como en los niveles internos de hoja y, dado que n es normalmente grande, la probabilidad de encontrar ciertos valores pronto es relativamente pequeña. Por otra parte, el hecho de que aparezcan menos claves de búsqueda en los nodos internos del árbol B, comparado con los árboles B⁺, implica que un árbol B tiene un grado de salida menor y, por tanto, puede que tenga una profundidad mayor que la correspondiente al árbol B⁺. Así, la búsqueda en un árbol B es más rápida para algunas claves de búsqueda pero más lenta para otras, aunque en general, el tiempo de la búsqueda es todavía proporcional al logaritmo del número de claves de búsqueda.

El borrado en un árbol B es más complicado. En un árbol B⁺ la entrada borrada siempre aparece en una hoja, mientras que en un árbol B, la entrada borrada podría aparecer en un nodo interno. El valor apropiado a colocar en su lugar se debe elegir del subárbol del nodo que contiene la entrada borrada. Concretamente, si se borra la clave de búsqueda K_i, la clave de búsqueda más pequeña que aparezca en el subárbol del puntero P_{i+1} se debe trasladar al campo ocupado anteriormente por K_i. Será necesario tomar otras medidas si ahora el nodo hoja tuviera pocas entradas. Por el contrario, la inserción en un árbol B es solo un poco más complicada que la inserción en un árbol B⁺.

Las ventajas de espacio que ofrecen los árboles B son escasas para índices grandes y normalmente no compensan los inconvenientes advertidos. De esta manera, la mayoría de implementadores de sistemas de bases de datos usan la estructura de datos de árbol B⁺, incluso aunque se refieran a la estructura (como se ha comentado antes) de datos como árbol B.

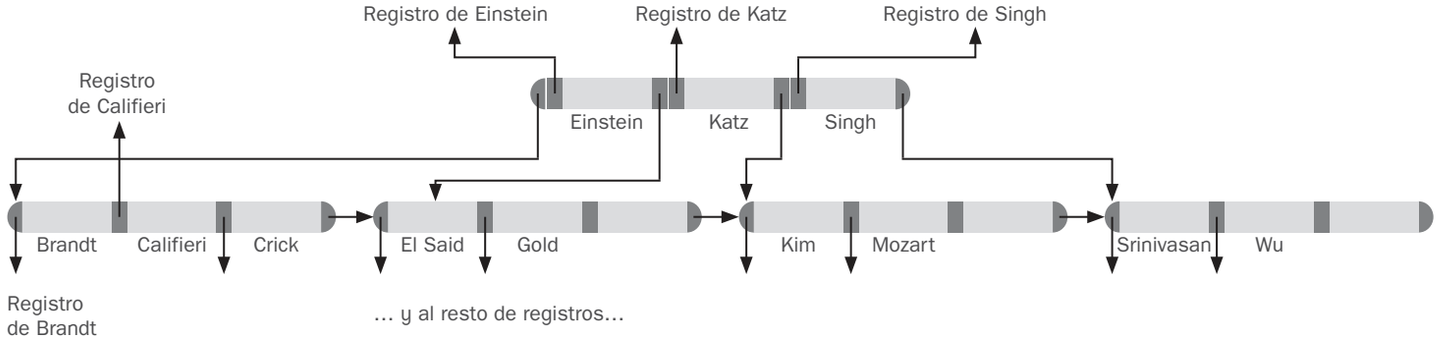


Figura 11.21. Árbol B equivalente al árbol B⁺ de la Figura 11.13.

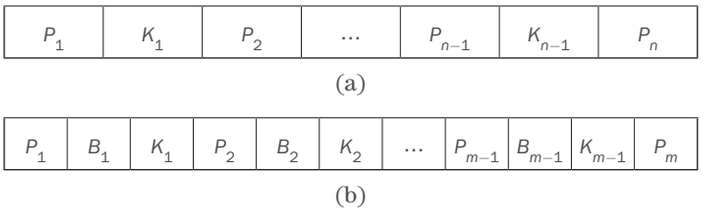


Figura 11.22. Nodos típicos de un árbol B. (a) Nodo hoja. (b) Nodo interno.

11.4.6. Memorias flash

En la descripción de la indexación hasta este momento, se ha supuesto que los datos residen en discos magnéticos. Aunque esta suposición sigue siendo cierta en la mayoría de los casos, la capacidad de las memorias flash ha crecido significativamente, y su coste por gigabyte ha caído de forma igualmente considerable, haciendo que en muchas aplicaciones el almacenamiento en memorias flash sea un serio competidor para sustituir al almacenamiento en discos magnéticos. Una pregunta natural es cómo afecta este cambio a la estructura de índices.

El almacenamiento en memorias flash se estructura en bloque y se puede usar la estructura de índices en árbol B^+ en esas memorias. Está clara la ventaja de un acceso mucho más rápido para la búsqueda de los índices. En lugar de necesitar un promedio de 10 milisegundos para buscar y leer un bloque, un bloque aleatorio se puede leer en aproximadamente un microsegundo desde una memoria flash. Por tanto, las búsquedas se ejecutarán significativamente de forma más rápida que con datos guardados en discos. El tamaño de nodo óptimo para el árbol B^+ en el caso de las memorias flash es normalmente más pequeño que para los discos.

La única desventaja real de las memorias flash es que no permiten actualizaciones en el mismo lugar de los datos en el nivel físico, aunque parezca que lógicamente sí lo hacen. Cada actualización se convierte en una copia + escritura de un bloque completo de la memoria flash, lo que requiere que la copia antigua del bloque tenga que borrarse; un borrado de un bloque tarda aproximadamente 1 milisegundo. Existen diversas investigaciones en marcha para desarrollar estructuras de índices que puedan reducir el número de borrados de bloques. Mientras tanto, los índices en árboles B^+ estándar se continuarán utilizando incluso con almacenamiento en memorias flash, con un rendimiento de actualización aceptable y con una mejora significativa respecto al almacenamiento en disco.

11.5. Accesos bajo varias claves

Hasta ahora se ha asumido implícitamente que solo se utiliza un índice sobre un atributo para procesar una consulta en una relación. Sin embargo, para ciertos tipos de consultas es ventajoso el uso de varios índices, si existen, o usar un índice construido sobre una clave de búsqueda de varios atributos.

11.5.1. Uso de varios índices de clave única

Suponga que el archivo *profesor* tiene dos índices: uno para el *nombre* y otro para *sueldo*. Dada la consulta: «Encontrar todas los profesores del departamento Finanzas con un sueldo igual a 80.000 €», se escribe:

select *ID*
from *profesor*
where *nombre_dept* ='Finanzas' **and** *sueldo* = 80000;

Hay tres estrategias posibles para procesar esta consulta:

1. Usar el índice *nombre_dept* para encontrar todos los registros correspondientes al departamento de Finanzas. Examinar luego esos registros para ver si *sueldo* = 80.000.
2. Usar el índice de *sueldo* para encontrar todos los registros pertenecientes a profesores con un sueldo de 80.000 €. Examinar luego esos registros para ver si el departamento es «Finanzas».
3. Usar el índice de *nombre_dept* para encontrar *punteros* a todos los registros pertenecientes al departamento de Finanzas. Usar también el índice de *sueldo* para encontrar los punteros a todos los registros correspondientes a profesores con un sueldo de 80.000 €. Los punteros que están en la intersección apuntan a la vez a los registros correspondientes a los profesores del departamento de Finanzas con un sueldo de 80.000 €.

La tercera estrategia es la única de las tres que aprovecha la ventaja de tener varios índices. Sin embargo, incluso esta estrategia podría ser una mala elección si se cumpliese lo siguiente:

- Existen muchos registros correspondientes al departamento de Finanzas.
- Existen muchos registros correspondientes a profesores con un sueldo de 80.000 €.
- Solo existen unos cuantos registros pertenecientes *tanto* al departamento de Finanzas como a profesores con un sueldo de 80.000 €.

Si se dieran estas tres condiciones se tendrían que examinar un gran número de punteros para generar un resultado reducido. La estructura de índices denominada «índice de mapas de bits» acelera significativamente la operación de intersección usada en la tercera estrategia. Los índices de mapas de bits se describen en la Sección 11.9.

11.5.2. Índices sobre varias claves

Una estrategia más eficiente para este caso es crear y utilizar un índice con una clave de búsqueda compuesta (*nombre_dept*, *sueldo*); es decir, la clave de búsqueda consistente en el nombre del departamento concatenado con el sueldo del profesor.

Se puede usar un índice ordenado (árbol B^+) para responder eficientemente a consultas de la forma:

select *ID*
from *profesor*
where *nombre_dept* = 'Finanzas' **and** *sueldo* = 80000;

Las consultas como la siguiente, que especifica una condición de igualdad sobre el primer atributo de la clave de búsqueda (*nombre_dept*) y un intervalo sobre el segundo (*sueldo*), también se pueden tratar eficientemente ya que corresponden a una consulta por intervalos sobre el atributo de búsqueda.

select *ID*
from *profesor*
where *nombre_dept* = 'Finanzas' **and** *sueldo* < 80000;

Incluso se puede utilizar un índice ordenado sobre la clave de búsqueda (*nombre_dept*, *sueldo*) para responder de manera eficiente a la siguiente consulta sobre un solo atributo:

select *ID*
from *profesor*
where *nombre_dept* = 'Finanzas';

La condición de igualdad *nombre_dept* = «Finanzas» es equivalente a una consulta por intervalos sobre el intervalo con extremo inferior (Finanzas, $-\infty$) y superior (Finanzas, $+\infty$). Las consultas por intervalos solo sobre el atributo *nombre_dept* se pueden tratar de forma similar.

Sin embargo, el uso de una estructura de índice ordenado con múltiples atributos presenta algunas deficiencias. Como ejemplo, considere la consulta:

select *ID*
from *profesor*
where *nombre_dept* < 'Finanzas' **and** *sueldo* < 80000;

Se puede responder a esta consulta usando un índice ordenado con la clave de búsqueda (*nombre_dept*, *sueldo*) de la manera siguiente: para cada valor de *nombre_dept* que está alfabéticamente antes que «Finanzas», hay que localizar los registros con un *sueldo* de 80.000 €. Sin embargo, debido a la ordenación de los registros en el archivo, es probable que cada registro esté en un bloque de disco diferente, lo que genera muchas operaciones de E/S.

La diferencia entre esta consulta y las dos anteriores es que la condición sobre el primer atributo (*nombre_dept*) es de comparación y no de igualdad. La condición no se corresponde con una consulta por intervalos sobre la clave de búsqueda.

Para acelerar el procesamiento en general de consultas con claves de búsqueda compuestas (que pueden implicar una o más operaciones de comparación) se pueden emplear varias estructuras especiales. En la Sección 11.9 se considerará la estructura de *índices de mapas de bits*. Existe otra estructura, denominada *árbol R*, que también se puede usar para este propósito. Los árboles *R* son una extensión de los árboles B^+ para el tratamiento de índices en varias dimensiones. Dado que se emplean fundamentalmente con datos de tipo geográfico, su estructura se describe en el Capítulo 25.

11.5.3. Índices de cobertura

Los **índices de cobertura** almacenan los valores de algunos atributos (distintos de los atributos de la clave de búsqueda) junto con los punteros a los registros. El almacenamiento de valores de atributo adicionales es útil en los índices secundarios, ya que permite responder algunas consultas utilizando únicamente el índice, sin ni siquiera buscar en los registros de datos.

Por ejemplo, supóngase un índice sin agrupación sobre el atributo *ID* de la relación *profesor*. Si se almacena el valor del atributo *sueldo* junto con el puntero a registro, se pueden responder consultas que soliciten el sueldo (sin el otro atributo, *nombre_dept*) sin acceder al registro de *profesor*.

Se obtendría el mismo efecto creando un índice sobre la clave de búsqueda (*ID*, *sueldo*), pero los índices de cobertura reducen el tamaño de las claves de búsqueda, lo que permite un mayor grado de salida en los nodos internos y puede reducir la altura del índice.

11.6. Asociación estática

Un inconveniente de la organización de archivos secuenciales es que hay que acceder a una estructura de índices para localizar los datos o utilizar una búsqueda binaria y, como resultado, más operaciones de E/S. La organización de archivos basada en la técnica de **asociación** (*hashing*) permite evitar el acceso a la estructura de índice. La asociación también proporciona una forma de construir índices. En los apartados siguientes se estudian las organizaciones de archivos y los índices basados en asociación.

En esta descripción de la asociación se usará el término **cajón** (*bucket*) para indicar una unidad de almacenamiento que puede guardar uno o más registros. Un cajón es normalmente un bloque de disco, aunque también se podría elegir de tamaño mayor o menor que un bloque de disco.

Formalmente, sea K el conjunto de todos los valores de clave de búsqueda y sea B el conjunto de todas las direcciones de cajón. Una **función de asociación** h es una función de K a B. Denominamos h a una función de asociación.

Para insertar un registro con clave de búsqueda K_i, se calcula $h(K_i)$, lo que proporciona la dirección del cajón para ese registro. De momento se supone que hay espacio en el cajón para almacenar el registro. Entonces, el registro se almacena en ese cajón.

Para realizar una búsqueda con el valor K_i de la clave de búsqueda, basta con calcular $h(K_i)$ y buscar luego el cajón con esa dirección. Supóngase que dos claves de búsqueda, K_5 y K_7, tienen el mismo valor de asociación; es decir, $h(K_5) = h(K_7)$. Si se realiza una búsqueda de K_5, el cajón $h(K_5)$ contendrá registros con valores de la clave de búsqueda K_5 y registros con valores de la clave de búsqueda K_7. Por tanto, hay que comprobar el valor de clave de búsqueda de todos los registros del cajón para asegurarse de que el registro es el deseado.

El borrado es igual de sencillo. Si el valor de la clave de búsqueda del registro que se debe borrar es K_i, se calcula $h(K_i)$, después se busca el cajón correspondiente a ese registro y se borra el registro del cajón.

La asociación se puede utilizar para dos propósitos diferentes. En la **organización de archivo asociativo** se obtiene directamente la dirección del bloque de disco que contiene el registro deseado calculando una función del valor de la clave de búsqueda del registro. En la **organización de índice asociativo** se organizan las claves de búsqueda con sus punteros asociados en una estructura de archivo asociativo.

11.6.1. Funciones de asociación

La peor función de asociación posible asigna todos los valores de la clave de búsqueda al mismo cajón. Una función así no es deseable, ya que hay que guardar todos los registros en el mismo cajón. Durante una búsqueda hay que examinar todos esos registros hasta encontrar el deseado. Una función de asociación ideal distribuye las claves almacenadas uniformemente entre todos los cajones, de modo que cada uno de ellos tenga el mismo número de registros.

Puesto que durante la etapa de diseño no se conocen con precisión los valores de la clave de búsqueda que se almacenarán en el archivo, lo deseable es elegir una función de asociación que asigne los valores de la clave de búsqueda a los cajones de manera que la distribución tenga las propiedades siguientes:

- Distribución *uniforme*. Es decir, la función de asociación asigna a cada cajón el mismo número de valores de la clave de búsqueda dentro del conjunto de *todos* los valores posibles de la misma.
- Distribución *aleatoria*. Es decir, en el caso promedio, cada cajón tendrá asignado casi el mismo número de valores, independientemente de la distribución real de los valores de la clave de búsqueda. Para ser más exactos, el valor de asociación no estará relacionado con ningún ordenamiento de los valores de la clave de búsqueda visible exteriormente como, por ejemplo, el orden alfabético o el orden determinado por la longitud de las claves de búsqueda; parecerá que la función de asociación es aleatoria.

Como ejemplo de estos principios se escogerá una función de asociación para el archivo *profesor* que utilice la clave búsqueda *nombre_dept*. La función de asociación que se escoja debe tener las propiedades deseadas no solo para el ejemplo del archivo *profesor* que se ha estado utilizando, sino también para un archivo *profesor* de tamaño real de una gran universidad con muchos departamentos.

Suponga que se decide tener 26 cajones y se define una función de asociación que asigna a los nombres que empiezan con la letra i-ésima del alfabeto el cajón i-ésimo. Esta función de asociación tiene la virtud de la simplicidad, pero no logra proporcionar una distribución uniforme, ya que se espera que haya más nombres que comiencen con letras como B y R que con Q y con X, por ejemplo.

Suponga ahora que se desea una función de asociación en la clave de búsqueda *sueldo*. Suponga que el sueldo mínimo es 30.000 € y el sueldo máximo es 130.000 € y se utiliza una función de asociación que divide el valor en 10 rangos, 30.000 €–40.000 €, 40.001 €–50.000 €, y así sucesivamente. La distribución de los valores de la clave de búsqueda es uniforme (ya que cada cajón tiene el mismo número de valores de *sueldo* diferentes), pero no es aleatoria. Los registros con sueldo entre 60.000 € y 70.000 € son más

frecuentes que los registros con sueldos entre 30.001 € y 40.000 €. En consecuencia, la distribución de los registros no es uniforme; algunos cajones reciben más registros que otros. Si la función presenta una distribución aleatoria, aunque se dieran esas correlaciones entre las claves de búsqueda, la aleatoriedad de la distribución haría, muy probablemente, que todos los cajones tuvieran más o menos el mismo número de registros, siempre y cuando cada clave de búsqueda apareciera solo en una pequeña parte de los registros (si una sola clave de búsqueda aparece en gran parte de los registros, es probable que el cajón que la contiene tenga más registros que otros cajones, independientemente de la función de asociación empleada).

cajón 0

cajón 1

15151	Mozart	Música	40000

cajón 2

32343	El Said	Historia	80000
58583	Califieri	Historia	60000

cajón 3

22222	Einstein	Física	95000
33456	Gold	Física	87000
98345	Kim	Electrónica	80000

cajón 4

12121	Wu	Finanzas	90000
76543	Singh	Finanzas	80000

cajón 5

76766	Crick	Biología	72000

cajón 6

10101	Srinivasan	Informática	65000
45565	Katz	Informática	75000
83821	Brandt	Informática	92000

cajón 7

Figura 11.23. Organización asociativa del archivo *profesor*, con *nombre_dept* como clave.

Las funciones de asociación habituales realizan cálculos sobre la representación binaria interna de la máquina de los caracteres de la clave de búsqueda. Una función de asociación sencilla de este tipo calcula en primer lugar la suma de las representaciones binarias de los caracteres de la clave y, luego, devuelve el *modulo* suma del número de cajones.

En la Figura 11.23 se muestra la aplicación de este esquema, con 10 cajones, al archivo *profesor*, con la suposición de que la letra i-ésima del alfabeto está representada por el número entero i.

La siguiente función de asociación es una alternativa mejor para la asociación de cadenas de caracteres. Sea s una cadena de caracteres de longitud n, y sea $s[i]$, el i-ésimo byte de la cadena. La función de asociación se define como:

$$s[0] * 31^{(n-1)} + s[1] * 31^{(n-2)} + \ldots + s[n-1]$$

La función se puede implementar eficientemente estableciendo el valor asociativo inicial como 0 e iterando desde el primero hasta el último carácter de la cadena, multiplicando en cada paso el valor asociativo por 31 y añadiendo el siguiente carácter (tratado como un entero). El resultado de aplicar la función anterior puede generar un número muy grande, pero realmente se calcula con un número fijo de enteros positivos; el resultado de cada multiplicación y suma se calcula, por tanto, como el valor *modulo* mayor de los enteros posibles más 1. El resultado de la función *modulo* del número de cajones se puede usar para la indexación.

Las funciones de asociación requieren un diseño cuidadoso. Una mala función de asociación puede hacer que la búsqueda tarde un tiempo proporcional al número de claves de búsqueda del archivo. Una función bien diseñada da un tiempo de búsqueda para casos promedio que es una constante (pequeña), independiente del número de claves de búsqueda del archivo.

11.6.2. Gestión de desbordamientos de cajones

Hasta ahora se ha asumido que cuando se inserta un registro el cajón al que se asigna tiene espacio para almacenarlo. Si el cajón no tiene suficiente espacio, se produce lo que se denomina **desbordamiento de cajones.** Los desbordamientos de cajones se pueden producir por varias razones:

- **Cajones insuficientes.** Se debe elegir un número de cajones, que se denota con n_B, tal que $n_B > n_r / f_r$, donde n_r indica el número total de registros que se van a almacenar y f_r indica el número de registros que caben en cada cajón. Esta designación, por supuesto, presupone que se conoce el número total de registros en el momento de definir la función de asociación.
- **Atasco.** Algunos cajones tienen asignados más registros que otros, por lo que algún cajón se puede desbordar, aunque otros cajones tengan todavía espacio libre. Esta situación se denomina **atasco** de cajones. El atasco puede producirse por dos motivos:
  1. Puede que varios registros tengan la misma clave de búsqueda.
  2. Puede que la función de asociación elegida genere una distribución no uniforme de las claves de búsqueda.

Para que la probabilidad de desbordamiento de cajones se reduzca, se escoge un número de cajones igual a $(n_r / f_r) * (1 + d)$, donde d es un factor de corrección, normalmente cercano a 0.2. Se pierde algo de espacio: alrededor del veinte por ciento del espacio de los cajones queda vacío. Pero la ventaja es que la probabilidad de desbordamiento se reduce.

Pese a la asignación de unos pocos cajones más de los necesarios, todavía se puede producir el desbordamiento de los cajones. El desbordamiento de los cajones se trata mediante **cajones de desbordamiento.** Si hay que insertar un registro en un cajón b y b está ya lleno, el sistema proporcionará un cajón de desbordamiento para b y el registro se insertará en ese cajón de desbordamiento. Si

el cajón de desbordamiento también se encuentra lleno, el sistema proporcionará otro cajón de desbordamiento, y así sucesivamente. Todos los cajones de desbordamiento de un cajón determinado se encadenan en una lista enlazada, como se muestra en la Figura 11.24. El tratamiento del desbordamiento mediante una lista enlazada se denomina **cadena de desbordamiento.**

Para tratar la cadena de desbordamiento es necesario modificar ligeramente el algoritmo de búsqueda. Como se dijo antes, el sistema utiliza la función de asociación sobre la clave de búsqueda para identificar un cajón *b*. El sistema debe examinar todos los registros del cajón *b* para ver si coinciden con la clave de búsqueda, como anteriormente. Además, si el cajón *b* tiene cajones de desbordamiento, también hay que examinar los registros de todos los cajones de desbordamiento.

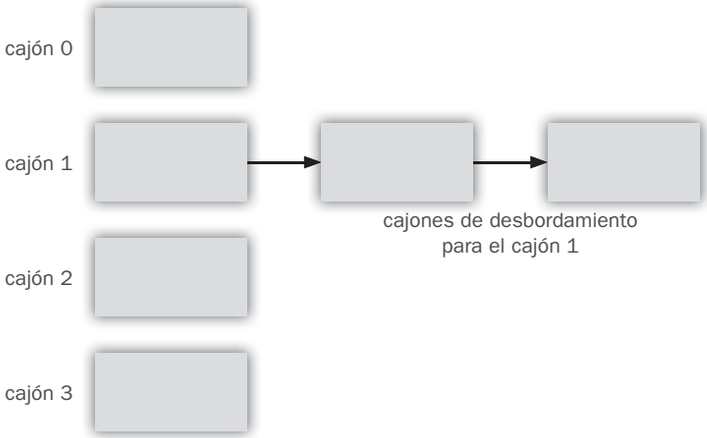


Figura 11.24. Cadena de desbordamiento en una estructura asociativa.

La forma de la estructura asociativa que se acaba de describir a veces se denomina **asociación cerrada.** En una aproximación alternativa, conocida como **asociación abierta,** se fija el conjunto de cajones y no hay cadenas de desbordamiento. En su lugar, si un cajón está lleno, el sistema inserta los registros en algún otro cajón del conjunto inicial *B* de cajones. Un criterio es utilizar el siguiente cajón (en orden cíclico) que tenga espacio; esta política se denomina *ensayo lineal*. Se utilizan también otros criterios, como el cálculo de funciones de asociación adicionales. La asociación abierta se emplea en la construcción de tablas de símbolos para compiladores y ensambladores, pero es preferible la asociación cerrada para los sistemas de bases de datos. El motivo es que el borrado con asociación abierta es problemático. Normalmente, los compiladores y los ensambladores solo realizan operaciones de búsqueda e inserción en sus tablas de símbolos. Sin embargo, en los sistemas de bases de datos es importante poder tratar tanto el borrado como la inserción. Por tanto, la asociatividad abierta solo tiene una importancia menor en la implementación de bases de datos.

Un inconveniente importante de la forma de asociación que se ha descrito es que hay que elegir la función de asociación cuando se implementa el sistema y no se puede cambiar fácilmente después si el archivo que se está indexando aumenta o disminuye de tamaño. Como la función *h* asigna valores de la clave de búsqueda a un conjunto fijo *B* de direcciones de cajón, si se hace *B* de gran tamaño se desperdicia espacio para manejar el futuro crecimiento del archivo. Si *B* es demasiado pequeño, cada cajón contendrá registros de muchos valores de la clave de búsqueda y se pueden provocar desbordamientos de cajones. A medida que el archivo aumenta de tamaño, el rendimiento se degrada. Más adelante, en la Sección 11.7, se estudiará la manera de cambiar dinámicamente el número de cajones y la función de asociación.

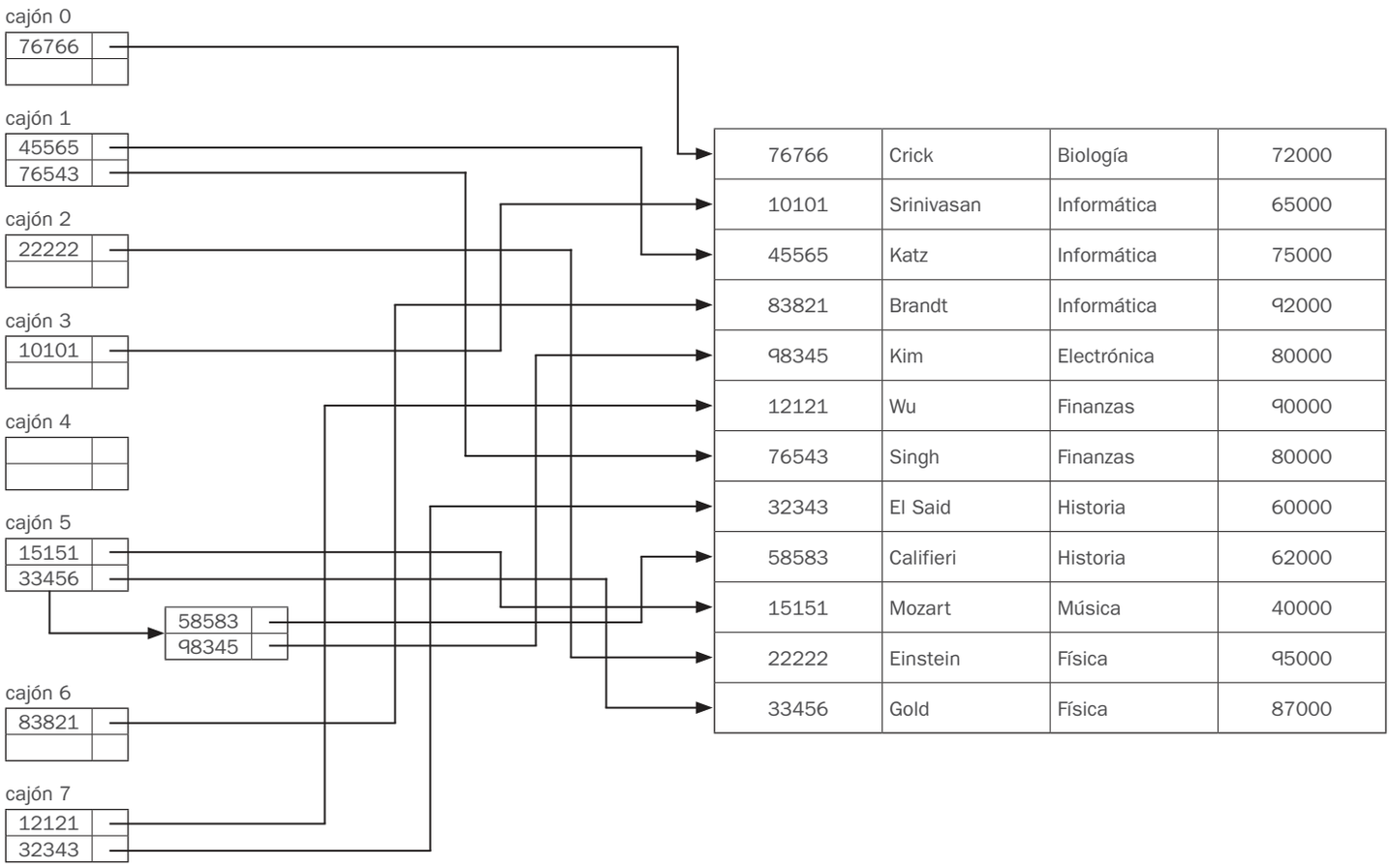


Figura 11.25. Índice asociativo de la clave de búsqueda *ID* del archivo *profesor*.

11.6.3. Índices asociativos

La asociatividad se puede utilizar no solamente para la organización de archivos, sino también para la creación de estructuras de índice. Un **índice asociativo** (*hash index*) organiza las claves de búsqueda, con sus punteros asociados, en una estructura de archivo asociativo. Los índices asociativos se construyen como se indica a continuación. Primero se aplica una función de asociación sobre la clave de búsqueda para identificar un cajón, y se almacenan la clave y los punteros asociados en el cajón (o en los cajones de desbordamiento). En la Figura 11.25 se muestra un índice asociativo secundario del archivo *profesor* para la clave de búsqueda *ID*. La función de asociación utilizada calcula la suma de las cifras del *ID modulo* ocho. El índice asociativo posee ocho cajones, cada uno de tamaño dos (un índice realista tendría, por supuesto, cajones de mayor tamaño). Uno de los cajones tiene tres claves asignadas, por lo que tiene un cajón de desbordamiento. En este ejemplo, *ID* es clave primaria de *profesor*, por lo que cada clave de búsqueda solo tiene asociado un puntero. En general, se pueden asociar varios punteros con cada clave.

Se usa el término *índice asociativo* para denotar las estructuras de archivo asociativo, así como los índices secundarios asociativos. Estrictamente hablando, los índices asociativos son solo estructuras de índices secundarios. Los índices asociativos no se necesitan nunca como estructuras de índices con agrupación ya que, si un archivo está organizado mediante asociatividad, no hay necesidad de ninguna estructura de índice asociativo adicional. Sin embargo, como la organización en archivos asociativos proporciona el mismo acceso directo a los registros que el indexado, se pretende que la organización de un archivo mediante asociación también tenga en él un índice con agrupación asociativo.

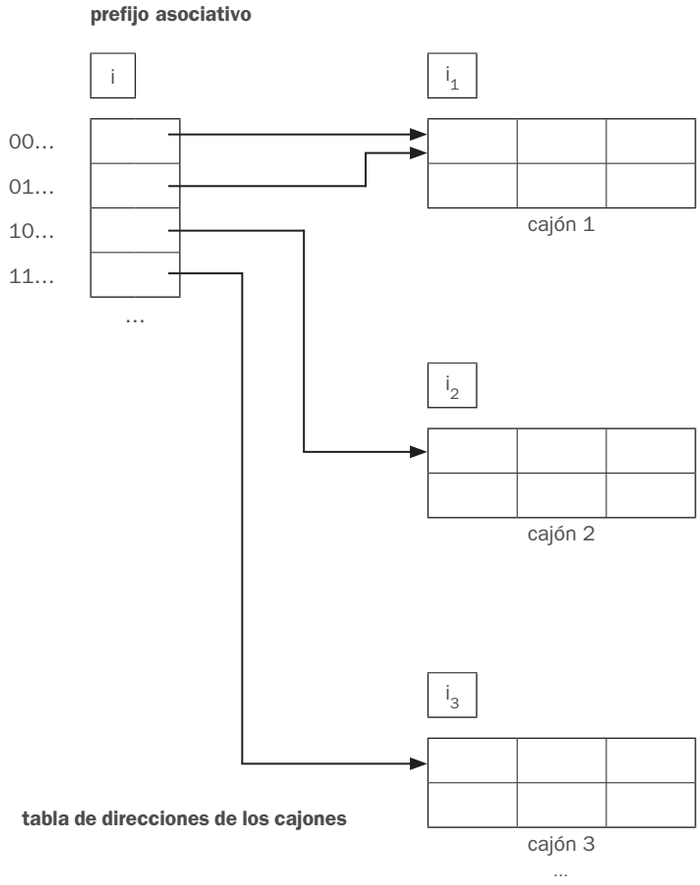


Figura 11.26. Estructura asociativa general extensible.

11.7. Asociación dinámica

Como se ha visto, la necesidad de fijar el conjunto B de direcciones de los cajones presenta un problema serio con la técnica de asociación estática vista en el apartado anterior. La mayor parte de las bases de datos aumentan de tamaño con el tiempo. Si se va a utilizar la asociación estática para esas bases de datos, hay tres opciones:

1. Elegir una función de asociación de acuerdo con el tamaño actual del archivo. Esta opción provocará una degradación del rendimiento a medida que la base de datos aumente de tamaño.
2. Elegir una función de asociación de acuerdo con el tamaño previsto del archivo en un momento determinado del futuro. Aunque se evite la degradación del rendimiento, puede que inicialmente se desperdicie una cantidad de espacio significativa.
3. Reorganizar periódicamente la estructura asociativa en respuesta al crecimiento del archivo. Esta reorganización supone elegir una nueva función de asociación, volver a calcular la función de asociación de cada registro del archivo y generar nuevas asignaciones de cajones. Esta reorganización es una operación masiva que consume mucho tiempo. Además, hay que prohibir el acceso al archivo durante la reorganización.

Algunas técnicas de **asociación dinámica** permiten modificar dinámicamente la función de asociación para adaptarse al aumento o disminución del tamaño de la base de datos. En este apartado se describe una forma de asociación dinámica, denominada **asociación extensible.** Las notas bibliográficas proporcionan referencias a otras formas de asociación dinámica.

11.7.1. Estructura de datos

La asociación extensible hace frente a los cambios del tamaño de la base de datos dividiendo y fusionando los cajones a medida que la base de datos aumenta o disminuye. En consecuencia, se conserva la eficiencia espacial. Además, puesto que la reorganización solo se lleva a cabo en un cajón simultáneamente, la degradación del rendimiento resultante es aceptablemente baja.

Con la asociación extensible se elige una función de asociación h con las propiedades deseadas de uniformidad y aleatoriedad. Sin embargo, esa función de asociación genera valores dentro de un rango relativamente amplio; por ejemplo, los enteros binarios de b bits. Un valor normal de b es 32.

No se crea un cajón para cada valor de la función de asociación. De hecho, 2^{32} es más de cuatro mil millones, y no son razonables tantos cajones salvo para las mayores bases de datos. En vez de eso, se crean cajones bajo demanda, a medida que se insertan registros en el archivo. Inicialmente no se utilizan todos los b bits del valor de la función de asociación. En cualquier momento se utilizan i bits, donde $0 \leq i \leq b$. Esos i bits son utilizados como reserva en una tabla adicional de direcciones de los cajones. El valor de i aumenta o disminuye con el tamaño de la base de datos.

En la Figura 11.26 se muestra una estructura general de asociación extensible. La i que aparece en la figura encima de la tabla de direcciones de los cajones indica que se requieren i bits del valor de la función de asociación $h(K)$ para determinar el cajón apropiado para K. Obviamente, ese número cambia a medida que el archivo aumenta de tamaño. Aunque se requieren i bits para encontrar la entrada correcta en la tabla de direcciones de los cajones, varias entradas consecutivas de la tabla pueden apuntar al mismo cajón. Todas esas entradas tendrán un prefijo de asociación común, pero la longitud de ese prefijo puede ser menor que i. Por lo tanto, se asocia con cada cajón un número entero que proporciona la longitud del prefijo de asociación común. En la Figura 11.26, el entero asociado con el cajón j aparece como i_j. El número de entradas de la tabla de direcciones de cajones que apuntan al cajón j es:

$$2^{(i-i_j)}$$

11.7.2. Consultas y actualizaciones

A continuación se estudiará la forma de realizar la búsqueda, la inserción y el borrado en una estructura asociativa extensible.

Para localizar el cajón que contiene el valor de la clave de búsqueda K_l, el sistema toma los primeros i bits más significativos de $h(K_l)$, busca la entrada de la tabla correspondiente a esa cadena de bits, y sigue el puntero del cajón de esa entrada de la tabla.

Para insertar un registro con un valor de clave de búsqueda K_l se sigue el mismo procedimiento de búsqueda que antes, y se llega a un cajón; por ejemplo, j. Si hay sitio en ese cajón, se inserta en él el registro. Si, por el contrario, el cajón está lleno, hay que dividir el cajón y redistribuir los registros actuales, junto con el nuevo. Para dividir el cajón, primero hay que determinar, a partir del valor de la función de asociación, si hace falta incrementar el número de bits que hay que utilizar.

- Si $i = i_j$, entonces solamente apunta al cajón j una entrada de la tabla de direcciones de los cajones. Por tanto, es necesario incrementar el tamaño de la tabla de direcciones de los cajones para incluir los punteros a los dos cajones que resultan de la división del cajón j. Esto se consigue considerando otro bit más del valor de asociación. Se incrementa en uno el valor de i, lo que duplica el tamaño de la tabla de direcciones de los cajones. Cada entrada se sustituye por dos entradas, ambas con el mismo puntero que la entrada original. Ahora, dos entradas de la tabla de direcciones de cajones apuntan al cajón j. Se asigna un nuevo cajón (el cajón z) y se hace que la segunda entrada apunte al nuevo cajón. Se definen i_j e i_z como i. A continuación se vuelve a calcular la función de asociación para todos los registros del cajón j y, en función de los primeros i bits (recuérdese que se ha añadido uno a i), se mantienen en el cajón j o se colocan en el cajón recién creado.

  Ahora se vuelve a intentar la inserción del nuevo registro. Normalmente el intento tiene éxito. Sin embargo, si todos los registros del cajón j, así como el registro nuevo, tienen el mismo prefijo de asociación, será necesario volver a dividir el cajón, ya que tanto los registros del cajón j como el registro nuevo tienen asignado el mismo cajón. Si la función de asociación se eligió cuidadosamente, es poco probable que una simple inserción provoque que un cajón se divida más de una vez, a menos que haya un gran número de registros con la misma clave de búsqueda. Si todos los registros del cajón j tienen el mismo valor de la clave de búsqueda, la división no servirá de nada. En esos casos se utilizan cajones de desbordamiento para almacenar los registros, como en la asociación estática.
- Si $i > i_j$, entonces más de una entrada en la tabla de direcciones de los cajones apunta al cajón j. Por tanto, se puede dividir el cajón j sin aumentar el tamaño de la tabla de direcciones de los cajones. Obsérvese que todas las entradas que apuntan al cajón j corresponden a prefijos de asociación que tienen el mismo valor en los i_j bits situados más a la izquierda. Se asigna un nuevo cajón (el cajón z) y se definen i_j e i_z con el valor que resulta de añadir uno al valor original de i_j. A continuación hay que ajustar las entradas de la tabla de direcciones de los cajones que anteriormente apuntaban al cajón j (observe que, con el nuevo valor de i_j, no todas las entradas corresponden a prefijos de asociación que tienen el mismo valor en los i_j bits situados más a la izquierda). La primera mitad de las entradas se deja como estaba (apuntando al cajón j) y se hace que el resto de las entradas apunten al cajón recién creado (el cajón z). Luego, como en el caso anterior, se vuelve a calcular la función de asociación para todos los registros del cajón j y se asignan o bien al cajón j o bien al cajón z recién creado.

  Luego se vuelve a intentar la inserción. En el improbable caso de que vuelva a fallar, se aplica uno de los dos casos, $i = i_j$ o $i > i_j$, según corresponda.

nombre_dept	h(*nombre_dept*)							
Biología	0010	1101	1111	1011	0010	1100	0011	0000
Informática	1111	0001	0010	0100	1001	0011	0110	1101
Electrónica	0100	0011	1010	1100	1100	0110	1101	1111
Finanzas	1010	0011	1010	0000	1100	0110	1001	1111
Historia	1100	0111	1110	1101	1011	1111	0011	1010
Música	0011	0101	1010	0110	1100	1001	1110	1011
Física	1001	1000	0011	1111	1001	1100	0000	0001

Figura 11.27. Función asociativa para *nombre_dept*.

Fíjese que en ambos casos solamente se necesita volver a calcular la función de asociación de los registros del cajón j.

Para borrar un registro con valor de la clave de búsqueda K_l se sigue el mismo procedimiento de búsqueda, que finaliza en un cajón; por ejemplo, j. Se borran tanto el registro del archivo como la clave de búsqueda del cajón. También se elimina el cajón si se queda vacío. Observe que, en este momento, se pueden fusionar varios cajones y se puede reducir el tamaño de la tabla de direcciones de los cajones a la mitad. El procedimiento para decidir los cajones que se deben fusionar y el momento de hacerlo se dejan al lector como ejercicio. Las condiciones bajo las que la tabla de direcciones de los cajones se puede reducir de tamaño también se dejan como ejercicio. A diferencia de la fusión de los cajones, el cambio de tamaño de la tabla de direcciones de los cajones es una operación bastante costosa si la tabla es de gran tamaño. Por tanto, solo merece la pena reducir el tamaño de la tabla de direcciones de los cajones si el número de cajones se reduce considerablemente.

Como ejemplo de la operación de inserción se usa el archivo *profesor* de la Figura 11.1 y se supone que la clave de búsqueda es *nombre_dept* con valores de asociación de 32 bits como se muestra en la Figura 11.27. Suponga que, inicialmente, el archivo está vacío, como se muestra en la Figura 11.28. Los registros se insertan de uno en uno. Para mostrar todas las características de la asociación extensible en una estructura pequeña, se hará la suposición no realista de que cada cajón solo puede contener dos registros.

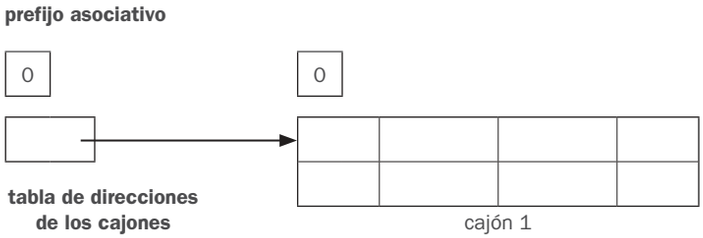


Figura 11.28. Estructura asociativa extensible inicial.

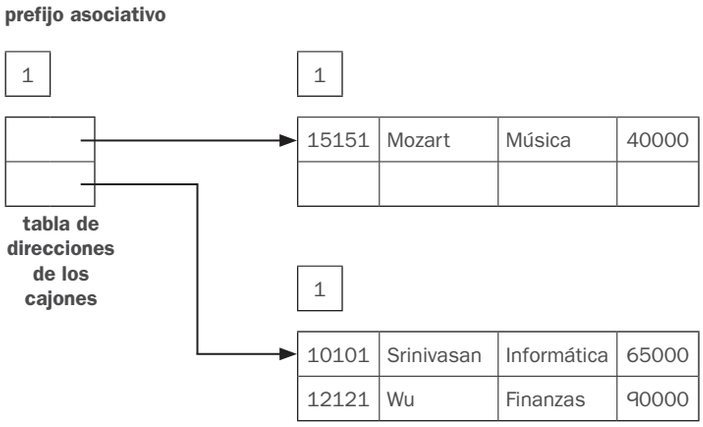


Figura 11.29. Estructura asociativa después de tres inserciones.

Se inserta el registro (10101, Srinivasan, Informática, 65000). La tabla de direcciones de los cajones contiene un puntero al único cajón existente y el sistema inserta el registro. A continuación se inserta el registro (12121, Wu, Finanzas, 90000). Este registro también se inserta en el único cajón de la estructura.

Cuando se intenta insertar el siguiente registro (15151, Mozart, Música, 40000), el cajón está lleno. Como $i = i_0$, es necesario incrementar el número de bits del valor de asociación que se utilizan. Ahora se utiliza un bit, lo que permite $2^1 = 2$ cajones. Este incremento en el número de bits necesarios requiere que se duplique el tamaño de la tabla de direcciones de cajones a dos entradas. El sistema divide el cajón, coloca en el nuevo aquellos registros cuya clave de búsqueda tiene un valor de asociación que comienza por 1 y deja el resto de los registros en el cajón original. En la Figura 11.29 se muestra el estado de la estructura después de la división.

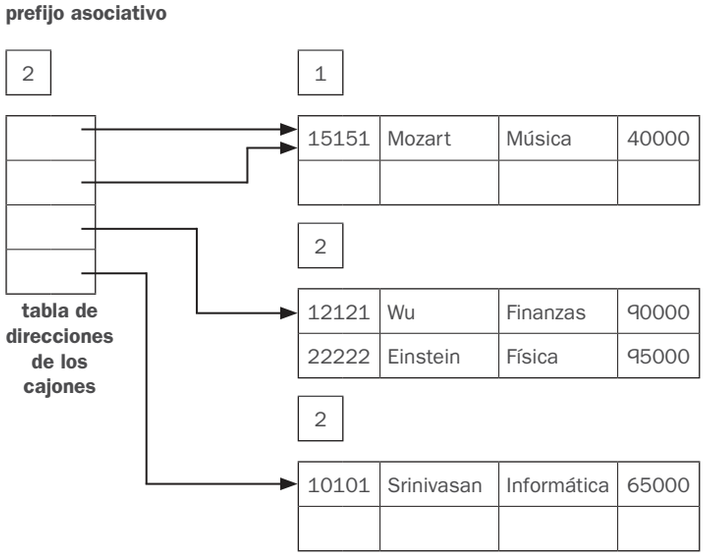


Figura 11.30. Estructura asociativa tras cuatro inserciones.

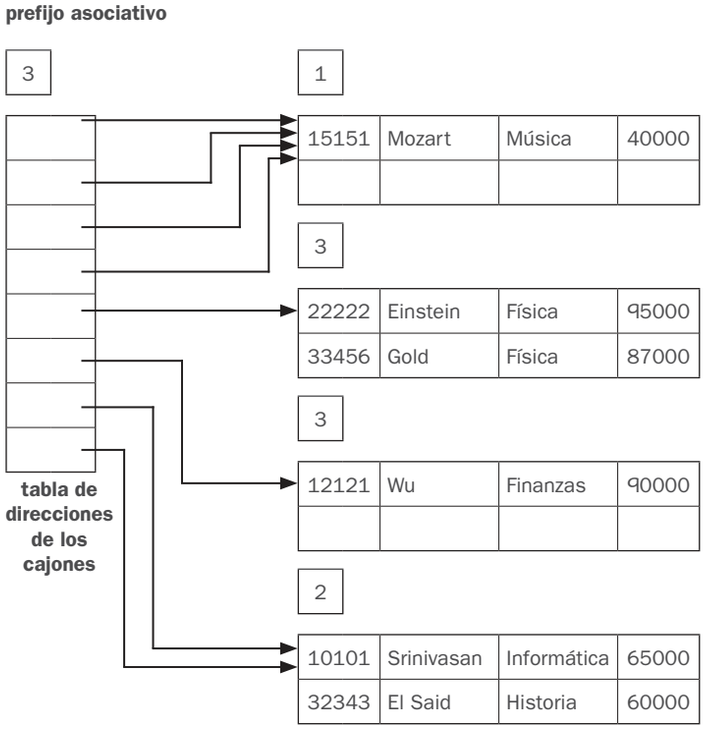


Figura 11.31. Estructura asociativa tras seis inserciones.

A continuación se inserta (22222, Einstein, Física, 95000). Como el primer bit de h(Física) es 1, hay que insertar ese registro en el cajón al que apunta la entrada «1» de la tabla de direcciones de los cajones. Una vez más, el cajón se encuentra lleno e $i = i_1$. Se incrementa a dos el número de bits del valor de asociación que se usan. Este incremento en el número de bits requiere que se duplique el tamaño de la tabla de direcciones de los cajones a cuatro entradas, como se muestra en la Figura 11.30. Como el cajón de la Figura 11.29 con el prefijo 0 del valor de asociación no se dividió, las dos entradas, 00 y 01, de la tabla de direcciones de los cajones apuntan a ese cajón.

Para cada registro del cajón de la Figura 11.29 con prefijo de asociación 1 (el cajón que se va a dividir), se examinan los dos primeros bits del valor de asociación para determinar el cajón de la nueva estructura que le corresponde.

A continuación se inserta el registro (32343, El Said, Historia, 60000), que se aloja en el mismo cajón que Informática. La siguiente inserción, la de (33456, Gold, Física, 87000), provoca un desbordamiento en un cajón, lo que causa el incremento del número de bits y la duplicación del tamaño de la tabla de direcciones de los cajones (véase la Figura 11.31).

La inserción del registro (45565, Katz, Informática, 75000) produce otro desbordamiento. Sin embargo, este desbordamiento se puede resolver sin incrementar el número de bits, ya que el cajón implicado tiene dos punteros apuntándole (véase la Figura 11.32).

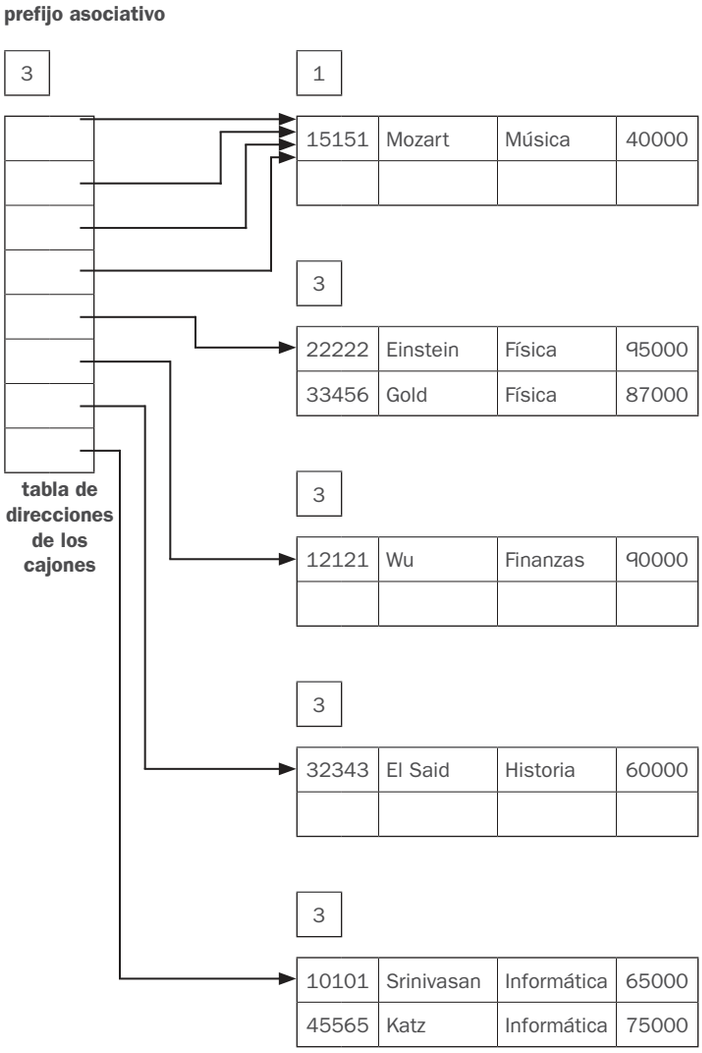


Figura 11.32. Estructura asociativa tras seis inserciones.

A continuación se insertan los registros de «Califieri», «Singh» y «Crick» sin desbordamiento de cajones. La inserción del tercer registro de Informática (83821, Brandt, Informática, 92000), produce un nuevo desbordamiento. Este desbordamiento no se puede resolver incrementando el número de bits ya que existen tres registros con exactamente el mismo valor de asociación. Por tanto, el sistema usa un cajón de desbordamiento, como se muestra en la Figura 11.33. Se continúa de esta forma hasta que se han insertado todos los registros de *profesor* de la Figura 11.1. La estructura final se puede ver en la Figura 11.34.

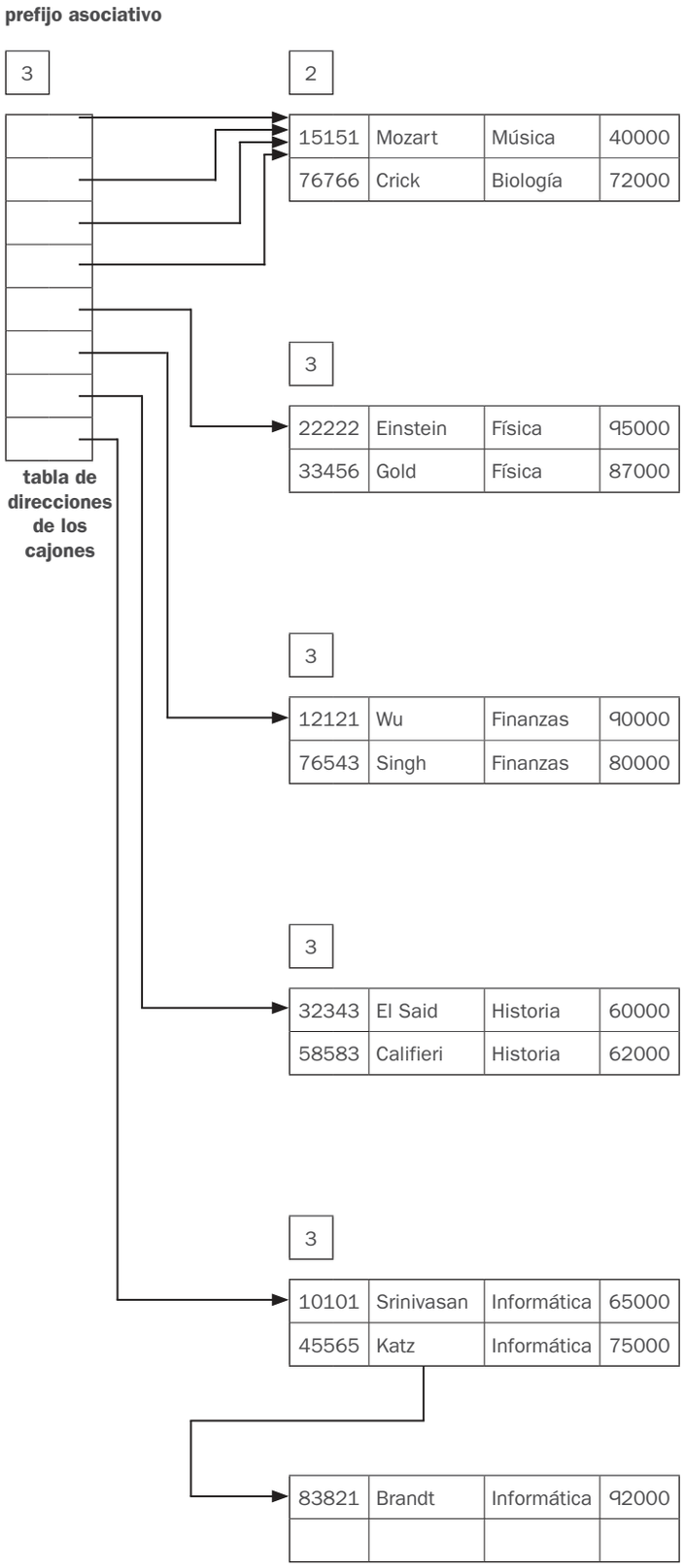


Figura 11.33. Estructura asociativa tras once inserciones.

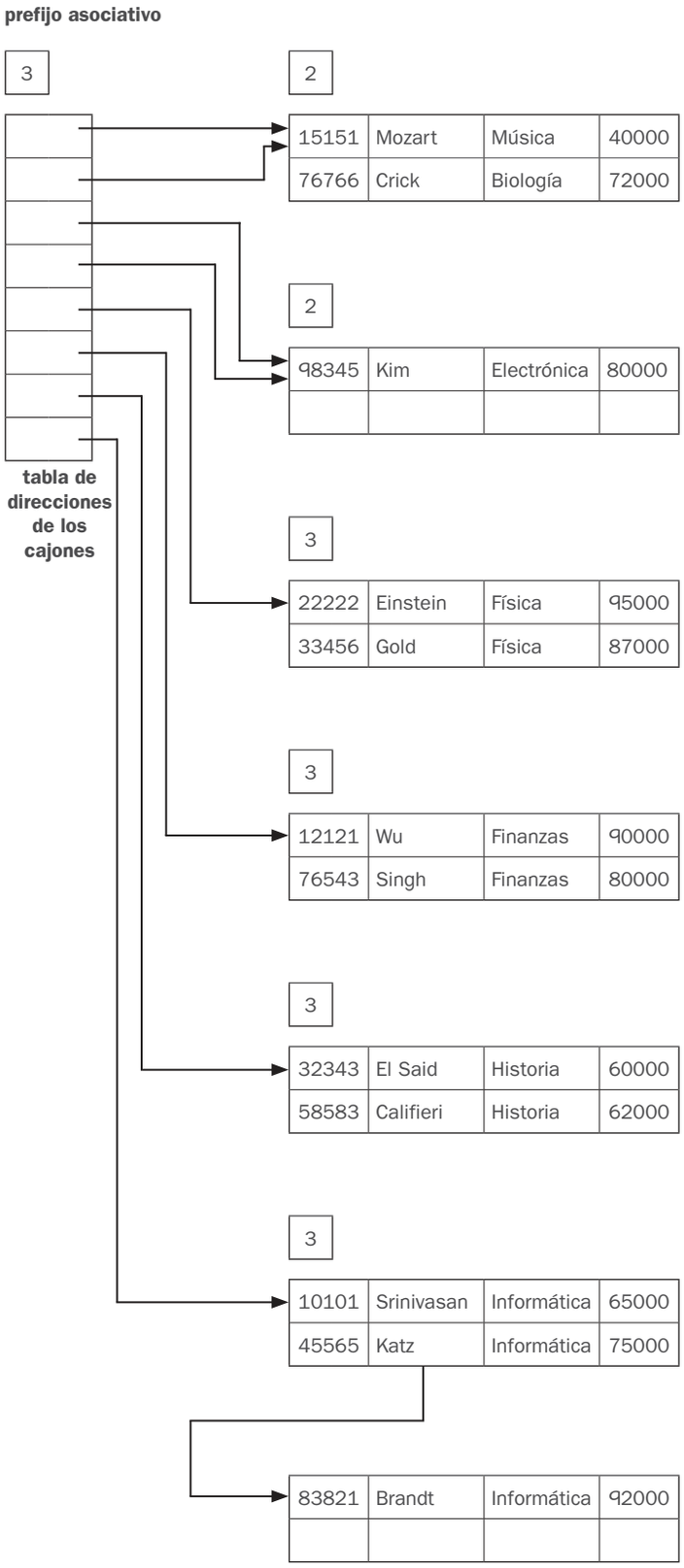


Figura 11.34. Estructura asociativa extensible para el archivo *profesor*.

11.7.3. Comparación entre la asociación estática y dinámica

A continuación se examinan las ventajas e inconvenientes de la asociación extensible respecto de la asociación estática. La ventaja principal de la asociación extensible es que el rendimiento no se degrada a medida que el archivo aumenta de tamaño. Además, el espacio adicional necesario es mínimo. Aunque la tabla de direcciones de los cajones provoca una sobrecarga adicional, solo contiene un puntero por cada valor de asociación para la longitud actual del prefijo. Por tanto, el tamaño de la tabla es pequeño. El principal ahorro de espacio de la asociación extensible respecto de otras formas de asociación es que no es necesario reservar cajones para un futuro aumento de tamaño; en vez de eso, los cajones se pueden asignar de manera dinámica.

Un inconveniente de la asociación extensible es que la búsqueda implica un nivel adicional de indirección, ya que se debe acceder a la tabla de direcciones de los cajones antes que a los propios cajones. Esta referencia adicional solo tiene una mínima repercusión en el rendimiento. Aunque las estructuras asociativas que se estudiaron en la Sección 11.6 no tienen este nivel adicional de referencia, cuando se llenan pierden esa mínima ventaja de rendimiento.

Por tanto, la asociación extensible se muestra como una técnica muy atractiva, siempre que se acepte la complejidad añadida de su implementación. En las notas bibliográficas se proporcionan descripciones más detalladas de la implementación de la asociación extensible.

Las notas bibliográficas también ofrecen referencias a otra forma de asociación dinámica denominada **asociación lineal,** que evita el nivel adicional de referencia asociado con la asociación extensible, con el coste adicional de más cajones de desbordamiento.

11.8. Comparación entre la indexación ordenada y la asociación

Se han examinado varios esquemas de indexación ordenada y varios esquemas de asociación. Se pueden organizar los archivos de registros como archivos ordenados, mediante una organización de índice secuencial u organizaciones de árbol B^+. Alternativamente, se pueden organizar los archivos mediante la asociación. Finalmente, los archivos se pueden organizar como montículos, en los que los registros no están ordenados de ninguna manera en especial.

Cada esquema tiene sus ventajas, dependiendo de la situación. Un fabricante de un sistema de bases de datos puede proporcionar muchos esquemas y dejar al diseñador de la base de datos la decisión final sobre los esquemas que se utilizarán. Sin embargo, este enfoque exige que el fabricante escriba más código, lo que aumenta tanto el coste del sistema como el espacio que ocupa. La mayor parte de los sistemas de bases de datos soportan árboles B^+ y pueden dar soporte también a alguna forma de organización de archivos asociativos o de índices asociativos.

Para hacer una buena elección de organización de archivos y de técnica de indexado, el fabricante o el diseñador de la base de datos deben tener en consideración los siguientes aspectos:

- ¿Es aceptable el coste de una reorganización periódica del índice o de la estructura asociativa?
- ¿Cuál es la frecuencia relativa de las inserciones y de los borrados?
- ¿Es deseable optimizar el tiempo medio de acceso a expensas de incrementar el tiempo de acceso en el caso peor?
- ¿Qué tipos de consultas se supone que van a formular los usuarios?

De estos puntos ya se han examinado los tres primeros, comenzando con la revisión de las ventajas relativas de las distintas técnicas de indexado, y nuevamente en la discusión de las técnicas de asociación. El cuarto punto, el tipo de consultas esperado, resulta fundamental para elegir entre la indexación ordenada y la asociación.

Si la mayoría de las consultas son de la forma:

select $A_1, A_2 \ldots, A_n$
from r
where $A_i = c$;

entonces, para procesarla, el sistema realiza una búsqueda en un índice ordenado o en una estructura asociativa del atributo A_i con el valor c. Para este tipo de consultas es preferible un esquema asociativo. Las búsquedas en índices ordenados requieren un tiempo proporcional al logaritmo del número de valores de A_i en r. Sin embargo, en las estructuras asociativas, el tiempo medio de búsqueda es una constante independiente del tamaño de la base de datos. La única ventaja de los índices respecto de las estructuras asociativas en este tipo de consultas es que el tiempo de búsqueda en el peor de los casos es proporcional al logaritmo del número de valores de A_i en r. Por el contrario, si se utiliza una estructura asociativa, el tiempo de búsqueda en el peor de los casos es proporcional al número de valores de A_i en r. Sin embargo, es poco probable en el caso de la asociación que se dé el peor caso de búsqueda posible (máximo tiempo de búsqueda) y, en este caso, es preferible emplear una estructura asociativa.

Las técnicas de índices ordenados son preferibles a las estructuras asociativas en los casos en que la consulta especifica un rango de valores. Estas consultas tienen el siguiente aspecto:

select $A_1, A_2 \ldots, A_n$
from r
where $A_i \leq c_2$ **and** $A_i \geq c_1$;

En otras palabras, la consulta anterior busca todos los registros con A_i valores comprendidos entre c_1 y c_2.

Considere la manera de procesar esta consulta empleando un índice ordenado. En primer lugar se realiza una búsqueda del valor c_1. Una vez que se ha encontrado el cajón que contiene el valor c_1, se sigue el orden de la cadena de punteros del índice para leer el siguiente cajón y se continúa de esta manera hasta que se llega a c_2.

Si en vez de un índice ordenado se tiene una estructura asociativa, se puede llevar a cabo una búsqueda de c_1 y localizar el cajón correspondiente; pero, en general, no resulta fácil determinar el cajón que hay que examinar a continuación. La dificultad surge porque una buena función de asociación asigna valores a los cajones de manera aleatoria. Por tanto, no hay un concepto sencillo de «siguiente cajón según el orden establecido». La razón por la que no se puede encadenar un cajón detrás de otro según un cierto orden de A_i es que cada cajón tiene asignado muchos valores de la clave de búsqueda. Como los valores están diseminados aleatoriamente según la función de asociación, es probable que los valores del intervalo especificado estén esparcidos por muchos cajones o, tal vez, por todos. Por esa razón, hay que leer todos los cajones para encontrar las claves de búsqueda necesarias.

Normalmente el diseñador elige la indexación ordenada, a menos que se sepa de antemano que las consultas de rango van a ser poco frecuentes, en cuyo caso se escoge la asociación. Las organizaciones asociativas resultan especialmente útiles para los archivos temporales creados durante el procesamiento de las consultas, siempre que se realicen búsquedas basadas en valores de la clave pero no se vayan a realizar consultas sobre intervalos.

11.9. Índices de mapas de bits

Los índices de mapas de bits son un tipo de índices especializados diseñados para una consulta sencilla sobre varias claves, aunque cada índice de mapas de bits se construya para una única clave.

Para utilizar índices de mapas de bits, los registros de la relación deben estar numerados secuencialmente comenzando, por ejemplo, desde 0. Dado un número n debe ser fácil recuperar el registro con número n. Esto resulta especialmente fácil de conseguir si los registros son de tamaño fijo y están asignados a bloques consecutivos de un archivo. El número de registro se puede traducir fácilmente en un número de bloque y en un número que identifica el registro dentro de ese bloque.

Considere una relación r con un atributo A que solo puede tomar como valor un número pequeño (por ejemplo, entre 2 y 20). Por ejemplo, la relación *info_profesor* puede tener el atributo *sexo*, que solo puede tomar los valores m (masculino) o f (femenino). Otro ejemplo puede ser el atributo *nivel_ingresos*, en el que los ingresos se han dividido en cinco niveles: $L1$: 0 € – 9999 €, $L2$: 10.000 € – 19.999 €, $L3$: 20.000 € – 39.999 €, $L4$: 40.000 € – 74.999 € y $L5$: 75.000 – ∞. Aquí, los datos originales pueden tomar muchos valores, pero un analista de datos los ha dividido en un número pequeño de rangos para simplificar su análisis.

11.9.1. Estructura de los índices de mapas de bits

Un **mapa de bits** es simplemente un *array* de bits. En su forma más sencilla, un **índice de mapas de bits** sobre el atributo A de la relación r consiste en un mapa de bits para cada valor que pueda tomar A. Cada mapa de bits tiene tantos bits como el número de registros de la relación. El i-ésimo bit del mapa de bits para el valor v_j se pone a 1 si el registro con número i tiene el valor v_j para el atributo A. El resto de los bits del mapa de bits se ponen a 0.

En este ejemplo hay un mapa de bits para el valor m y otro para f. El i-ésimo bit del mapa de bits para m se pone a 1 si el valor *sexo* del registro con número i es m. El resto de bits del mapa de bits de m se ponen a 0. Análogamente, el mapa de bits de f tiene el valor 1 para los bits correspondientes a los registros con el valor f para el atributo *sexo*; el resto de bits tienen el valor 0. La Figura 11.35 muestra un ejemplo de índices de mapa de bits para la relación *info_profesor*.

número de registro	*ID*	*sexo*	*nivel_ingresos*
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Mapas de bits para *sexo*

m	10010
f	01101

Mapas de bits para *nivel_ingresos*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Figura 11.35. Índices de mapas de bits para la relación *info_profesor*.

Ahora se considerará cuándo resultan útiles los mapas de bits. La manera más sencilla de recuperar todos los registros con el valor m (o f) sería simplemente leer todos los registros de la relación y seleccionar los de valor m (o f, respectivamente). El índice de mapas de bits no ayuda realmente a acelerar esa selección. Aunque nos permitiría leer solo los registros para un determinado sexo, es probable que haya que leer, de todas formas, todos los bloques del disco.

De hecho, los índices de mapas de bits resultan útiles para las selecciones sobre todo cuando hay selecciones bajo varias claves. Suponga que se crea un índice de mapas de bits sobre el atributo *nivel_ingresos*, que ya se ha descrito antes, además del índice de mapas de bits para *sexo*.

Considere ahora una consulta que seleccione mujeres con ingresos en el rango 10.000 € –19.999 €. Esta consulta se puede expresar como:

select *
from r
where *sexo* = 'f' **and** *nivel_ingresos* = 'L2';

Para evaluar esta selección se busca el valor f en el mapa de bits de *sexo* y el valor $L2$ en el de *nivel_ingresos* y se realiza la **intersección** (conjunción lógica) de los dos mapas de bits. En otras palabras, se calcula un nuevo mapa de bits en el que el bit i tenga el valor 1 si el i-ésimo bit de los dos mapas de bits es 1, y el valor 0 en caso contrario. En el ejemplo de la Figura 11.35, la intersección del mapa de bits de *sexo* = f (01101) y el de *nivel_ingresos* = $L2$ (01000) da como resultado el mapa de bits 01000.

Como el primer atributo puede tomar dos valores y el segundo cinco, se puede esperar, en promedio, que solo 1 de cada 10 registros satisfaga la condición combinada de los dos atributos. Si hay más condiciones, es probable que la proporción de los registros que satisfacen todas las condiciones sea bastante pequeña. El sistema puede calcular así el resultado de la consulta buscando todos los bits con valor 1 del mapa de bits resultado de la intersección y recuperando los registros correspondientes. Si la proporción es grande, la exploración de la relación completa seguirá siendo la alternativa menos costosa.

Otro uso importante de los mapas de bits es el recuento del número de tuplas que satisfacen una selección dada. Esas consultas son importantes para el análisis de datos. Por ejemplo, si se desea determinar el número de mujeres que tienen un nivel de ingresos $L2$, se calcula la intersección de los dos mapas de bits y luego se cuenta el número de bits de valor 1 en el mapa de bits resultado de la intersección. Así se puede obtener el resultado deseado del índice de mapa de bits sin ni siquiera acceder a la relación.

Los índices de mapas de bits son generalmente bastante pequeños en comparación con el tamaño real de la relación. Los registros suelen tener entre decenas y centenares de bytes de longitud, mientras que un solo bit representa a un registro en el mapa de bits. Por tanto, el espacio ocupado por cada mapa de bits suele ser menos del uno por ciento del espacio ocupado por la relación. Por ejemplo, si el tamaño del registro de una relación dada es de 100 bytes, el espacio ocupado por cada mapa de bits sería la octava parte del uno por ciento del espacio ocupado por la relación. Si el atributo A de la relación solo puede tomar un valor de entre ocho, el índice de mapas de bits de ese atributo consiste en ocho mapas de bits que, juntos, solo ocupan el uno por ciento del tamaño de la relación.

El borrado de registros crea huecos en la secuencia de registros, ya que el desplazamiento de registros (o de los números de registro) para rellenar los huecos resultaría excesivamente costoso. Para reconocer los registros borrados se puede almacenar un **mapa de bits de existencia** en el que el bit i sea 0 si el registro i no existe, y 1 en caso contrario. Se verá la necesidad de los mapas de bits de existencia en la Sección 11.9.2. La inserción de registros no debe afectar a la secuencia de numeración de los demás registros. Por tanto, se puede insertar tanto añadiendo registros al final del archivo como reemplazando los registros borrados.

11.9.2. Implementación eficiente de las operaciones de mapas de bits

Se puede calcular fácilmente la intersección de dos mapas de bits usando un bucle **for:** la iteración i-ésima del bucle calcula la conjunción **(and)** de los bits i-ésimos de los dos mapas de bits. Se puede acelerar considerablemente el cálculo de la intersección usando la instrucción **and** de bits del que disponen la mayoría de los conjuntos de instrucciones de las computadoras. Cada *palabra* suele constar de 32 o 64 bits, en función de la arquitectura de la computadora. La instrucción de bits **and** toma dos palabras como entrada y devuelve una palabra en la que cada bit es la conjunción lógica de los bits de igual posición de las palabras de entrada. Lo que es importante observar es que una sola instrucción de bits **and** puede calcular la intersección de 32 o de 64 bits *a la vez*.

Si una relación tuviese un millón de registros, cada mapa de bits contendría un millón de bits o, lo que es equivalente, 128 kilobytes. Solo se necesitan 31.250 instrucciones para calcular la intersección de dos mapas de bits de la relación, suponiendo un tamaño de palabra de 32 bits. Por tanto, el cálculo de intersecciones de mapas de bits es una operación extremadamente rápida.

Al igual que la intersección de mapas de bits resulta útil para calcular la conjunción de dos condiciones, la unión de mapas de bits resulta útil para calcular la disyunción **(or)** de dos condiciones. El procedimiento para la unión de mapas de bits es exactamente igual que el de la intersección, salvo que se utiliza la instrucción de bits **or** en lugar de **and.**

La operación complemento se puede usar para calcular un predicado que incluya la negación de una condición, como **not** ($nivel_ingresos = L1$). El complemento de un mapa de bits se genera complementando cada uno de sus bits (el complemento de 1 es 0 y el complemento de 0 es 1). Puede parecer que **not** ($nivel_ingresos = L1$) se puede implementar simplemente calculando el complemento del mapa de bits del nivel de ingresos $L1$. Sin embargo, si se ha borrado algún registro, el mero cálculo del complemento del mapa de bits no resulta suficiente. Los bits correspondientes a esos registros serán 0 en el mapa de bits original, pero pasarán a ser 1 en el complemento, aunque el registro no exista. También surge un problema similar cuando el valor de un atributo es *nulo*. Por ejemplo, si el valor de $nivel_ingresos$ es nulo, el bit será 0 en el mapa de bits original para el valor $L1$ y 1 en el complementado.

Para asegurarse de que los bits correspondientes a registros borrados se ponen a 0 en el resultado, hay que realizar la intersección del mapa de bits complementado con el mapa de bits de existencia para desactivar los bits de los registros borrados. Análogamente, para manejar los valores nulos, también se debe realizar la intersección del mapa de bits complementado con el complemento del mapa de bits para el valor *nulo*.[2]

Se puede contar rápidamente el número de bits que valen 1 en el mapa de bits si se emplea una técnica inteligente. Se puede mantener un *array* de 256 entradas, donde la entrada i-ésima almacene el número de bits que valen 1 en la representación binaria de i. Hay que definir el recuento inicial como 0. Se toma cada byte del mapa de bits, se usa para indexar en el *array* y se añade el recuento almacenado al recuento total. El número de operaciones de suma sería la octava parte del número de tuplas y, por tanto, el proceso de recuento es muy eficiente. Un gran *array* (que emplee $2^{16} = 65.536$ entradas), indexado por pares de bytes, dará incluso aceleraciones mayores, pero con un coste de almacenamiento superior.

2 El tratamiento de predicados como **is unknown** puede causar aún más complicaciones, que requerirían en general el uso de un mapa de bits adicional para determinar los resultados de las operaciones que son desconocidos.

11.9.3. Mapas de bits y árboles B+

Los mapas de bits se pueden combinar con los índices normales de árboles B+ para las relaciones donde unos pocos valores de los atributos sean extremadamente frecuentes y también aparezcan otros valores, pero con mucha menor frecuencia. En las hojas de los índices de los árboles B+, para cada valor se suele mantener una lista de todos los registros con ese valor para el atributo indexado. Cada elemento de la lista sería un identificador de registro que consta, al menos, de 32 bits, y normalmente más. Para cada valor que aparece en muchos registros se almacena un mapa de bits en lugar de una lista de registros.

Suponga que un valor dado v_i aparece en la dieciseisava parte de los registros de una relación. Sea N el número de registros de la relación y supóngase que cada registro tiene un número de 64 bits que lo identifica. El mapa de bits solo necesita un bit por registro, o N en total. En cambio, la representación de lista necesita 64 bits por registro en el que aparezca el valor, o $64 * N/16 = 4N$ bits. Por tanto, es preferible el mapa de bits para representar la lista de registros del valor v_i. En el ejemplo (con un identificador de registros de 64 bits), si tiene un valor dado inferior a uno de cada sesenta y cuatro registros, es preferible la representación de lista de registros para la identificación de los registros con ese valor, ya que emplea menos bits que la representación con mapas de bits. Si más de uno de cada sesenta y cuatro registros tiene ese valor dado, es preferible la representación de mapas de bits.

Por tanto, los mapas de bits se pueden emplear como mecanismo de almacenamiento comprimido en los nodos hoja de los árboles B+ de los valores que aparecen muy frecuentemente.

11.10. Definición de índices en SQL

La norma SQL no proporciona al usuario o administrador de la base de datos ninguna forma de controlar qué índices se crean y se mantienen por el sistema de base de datos. Los índices no se necesitan para la corrección, ya que son estructuras de datos redundantes. Sin embargo, los índices son importantes para el procesamiento eficiente de las transacciones, incluyendo las transacciones de actualización y consulta. Los índices son también importantes para conseguir un cumplimiento eficiente de las restricciones de integridad.

En principio, un sistema de base de datos puede decidir automáticamente qué índices crear. Sin embargo, debido al coste en espacio de los índices, así como al efecto de los índices en el procesamiento de actualizaciones, no es fácil hacer automáticamente una elección apropiada sobre qué índices mantener. Por este motivo, la mayoría de las implementaciones de SQL proporcionan al programador control sobre la creación y eliminación de índices mediante comandos del lenguaje de definición de datos.

A continuación se muestra la sintaxis de estos comandos. Aunque la sintaxis que se muestra se usa ampliamente y existe en muchos sistemas de bases de datos, no es parte de la norma SQL. La norma SQL no da soporte al control del esquema físico de la base de datos; se restringe al esquema lógico de la base de datos.

Un índice se crea mediante el comando **create index,** que tiene la forma:

create index <nombre-índice> **on** <nombre-relación> (<lista-atributos>);

La *lista-atributos* es la lista de atributos de la relación que constituye la clave de búsqueda del índice.

Para definir un índice llamado *índice_dept* de la relación *profesor* con la clave de búsqueda *nombre_dept*, se escribe:

create index *índice_dept* **on** *profesor* (*nombre_dept*);

Si se desea declarar que la clave de búsqueda es una clave candidata hay que añadir el atributo **unique** a la definición del índice. Con esto, el comando:

create unique index *índice_dept* **on** *profesor* (*nombre_dept*);

declara *nombre_dept* como una clave candidata de *profesor* (lo que no es probable que queramos realmente para la base de datos de la universidad). Si cuando se introduce el comando **create unique index,** *nombre_dept* no es una clave candidata, se mostrará un mensaje de error y fallará el intento de crear el índice. Por otro lado, si el índice se crea correctamente, fallará cualquier intento de insertar una tupla que viole la declaración de clave. Fíjese que la característica **unique** es redundante si el sistema de bases de datos soporta la declaración **unique** de SQL estándar.

Muchos sistemas de bases de datos ofrecen un modo de especificar el tipo de índice que se va a utilizar (como los árboles B^+ o la asociación). Algunos sistemas de bases de datos permiten también que se declare uno de los índices de una relación como agrupado; el sistema almacena entonces la relación ordenada de acuerdo con la clave de búsqueda del índice agrupado.

Para hacer posible la eliminación (drop) de un índice es necesario especificar su nombre. El comando **drop index** tiene la forma:

drop index <nombre-índice>;

11.11. Resumen

- Muchas consultas solamente hacen referencia a una pequeña proporción de los registros de un archivo. Para reducir la sobrecarga en la búsqueda de estos registros se pueden construir *índices* para los archivos almacenados en la base de datos.
- Los archivos secuenciales indexados son unos de los esquemas de índice más antiguos usados en los sistemas de bases de datos. Para permitir una rápida recuperación de los registros según el orden de la clave de búsqueda, los registros se almacenan consecutivamente y los que no siguen el orden se encadenan entre sí. Para permitir un acceso aleatorio, se usa una estructura índice.
- Existen dos tipos de índices que se pueden utilizar: los índices densos y los índices dispersos. Los índices densos contienen una entrada por cada valor de la clave de búsqueda, mientras que los índices dispersos contienen entradas solo para algunos de esos valores.
- Si el orden de una clave de búsqueda se corresponde con el orden de una relación, un índice sobre la clave de búsqueda se conoce como *índice con agrupación*. Los otros índices son los *índices sin agrupación* o *secundarios*. Los índices secundarios mejoran el rendimiento de las consultas que utilizan otras claves de búsqueda distintas de las del índice con agrupación. Sin embargo, estas implican una sobrecarga adicional en la modificación de la base de datos.
- El inconveniente principal de la organización de archivo secuencial indexado es que el rendimiento disminuye según crece el archivo. Para superar esta deficiencia se puede usar un *índice de árbol* B^+.
- Un índice de árbol B^+ tiene la forma de un árbol *equilibrado*, en el cual cada camino de la raíz a las hojas del árbol tiene la misma longitud. La altura de un árbol B^+ es proporcional al logaritmo en base N del número de registros de la relación, donde cada nodo interno almacena N punteros; el valor de N está usualmente entre 50 y 100. Los árboles B^+ son más cortos que otras estructuras de árboles binarios equilibrados como los árboles AVL y, por tanto, necesitan menos accesos a disco para localizar los registros.
- Las búsquedas en un índice de árbol B^+ son directas y eficientes. Sin embargo, la inserción y el borrado son algo más complicados, pero eficientes. El número de operaciones que se necesitan para la inserción y borrado en un árbol B^+ es proporcional al logaritmo en base N del número de registros de la relación, donde cada nodo interno almacena N punteros.
- Se pueden utilizar los árboles B^+ tanto para indexar un archivo con registros como para organizar los registros de un archivo.
- Los índices de árbol B son similares a los índices de árbol B^+. La mayor ventaja de un árbol B es que el árbol B elimina el almacenamiento redundante de los valores de la clave de búsqueda. Los inconvenientes principales son la complejidad y el reducido grado de salida para un tamaño de nodo dado. En la práctica, los diseñadores de sistemas prefieren, casi universalmente, los índices de árbol B^+.
- Las organizaciones de archivos secuenciales necesitan una estructura de índice para localizar los datos. Los archivos con organizaciones basadas en asociación, en cambio, permiten encontrar la dirección de un elemento de datos directamente mediante el cálculo de una función con el valor de la clave de búsqueda del registro deseado. Como a la hora de diseñar no se sabe la manera precisa en la cual los valores de la clave de búsqueda se van a almacenar en el archivo, una buena función de asociación es la que distribuya los valores de la clave de búsqueda en los cajones de una manera uniforme y aleatoria.
- La *asociación estática* utiliza una función de asociación en la que el conjunto de direcciones de cajones está fijado. Estas funciones de asociación no se pueden adaptar fácilmente a las bases de datos que tengan un crecimiento significativo con el tiempo. Hay varias *técnicas de asociación dinámica* que permiten que la función de asociación cambie. Un ejemplo es la *asociación extensible*, que trata los cambios de tamaño de la base datos mediante la división y fusión de cajones según crezca o disminuya la base de datos.
- También se puede utilizar la asociación para crear índices secundarios; tales índices se denominan *índices asociativos*. Por motivos de notación se asume que las organizaciones de archivos asociativos tienen un índice asociativo implícito sobre la clave de búsqueda de la asociación.
- Los índices ordenados con árboles B^+ y con índices asociativos se pueden usar para la selección basada en condiciones de igualdad que involucren atributos únicos. Cuando hay varios atributos en una condición de selección se puede realizar la intersección de los identificadores de los registros recuperados con los diferentes índices.
- Los índices de mapas de bits proporcionan una representación muy compacta para la indexación de atributos con muy pocos valores distintos. Las operaciones de intersección son extremadamente rápidas en los mapas de bits, haciéndolos ideales para realizar consultas sobre varios atributos.

Términos de repaso

- Tipos de acceso.
- Tiempo de acceso.
- Tiempo de inserción.
- Tiempo de borrado.
- Espacio adicional.
- Índice ordenado.
- Índice con agrupación.
- Índice primario.
- Índice sin agrupación.
- Índice secundario.
- Archivo secuencial indexado.
- Registro/entrada del índice.
- Índice denso.
- Índice disperso.
- Índice multinivel.
- Clave compuesta.
- Exploración secuencial.
- Índice de árbol B^+.
- Nodo hoja.
- Nodo interno.
- Árbol equilibrado.
- Consulta de intervalo.
- División de nodo.
- Fusión de nodos.
- Clave de búsqueda no única.
- Organización de archivos con árboles B^+.
- Carga en bruto.
- Construcción de árbol B^+ de abajo arriba.
- Índice de árbol B.
- Asociación estática.
- Organización de archivos asociativos.
- Índice asociativo.
- Cajón.
- Función de asociación.
- Desbordamiento de cajones.
- Atasco.
- Asociación cerrada.
- Asociación dinámica.
- Asociación extensible.
- Acceso bajo varias claves.
- Índices sobre varias claves.
- Índice de mapas de bits.
- Operaciones de mapas de bits.
  - Intersección.
  - Unión.
  - Complemento.
  - Mapa de bits de existencia.

Ejercicios prácticos

11.1. Los índices agilizan el procesamiento de consultas, pero normalmente es una mala idea crear índices para todos los atributos y para todas las combinaciones de atributos que sean potenciales claves de búsqueda. Explique por qué.

11.2. ¿Es posible en general tener dos índices con agrupación en la misma relación para dos claves de búsqueda diferentes? Razone la respuesta.

11.3. Construya un árbol B^+ con el siguiente conjunto de valores de la clave:

$$(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)$$

Suponga que el árbol está inicialmente vacío y que se añaden los valores en orden ascendente. Construya árboles B^+ para los casos en los que el número de punteros que caben en un nodo son:

a. Cuatro.

b. Seis.

c. Ocho.

11.4. Para cada árbol B^+ del Ejercicio práctico 11.3 muestre el aspecto del árbol después de cada una de las siguientes operaciones:

a. Insertar 9.

b. Insertar 10.

c. Insertar 8.

d. Borrar 23.

e. Borrar 19.

11.5. Considere el esquema modificado de redistribución para árboles B^+ que se describe en la segunda parte de la Sección 11.4.1. ¿Cuál es la altura esperada del árbol en función de n?

11.6. Suponga que se está usando la asociación extensible en un archivo que contiene registros con los siguientes valores de la clave de búsqueda:

$$2, 3, 5, 7, 11, 17, 19, 23, 29, 31$$

Dibuje la estructura asociativa extensible para este archivo si la función de asociación es $h(x) = x \bmod 8$ y los cajones pueden contener hasta tres registros.

11.7. Muestre cómo cambia la estructura asociativa extensible del Ejercicio práctico 11.6 como resultado de realizar los siguientes pasos:

a. Borrar 11.

b. Borrar 31.

c. Insertar 1.

d. Insertar 15.

11.8. Escriba el pseudocódigo para la función de árboles B^+ buscarIterador(), que es como la función buscar(), excepto que devuelve un objeto iterador como se describe en la Sección 11.3.2. Escriba también el pseudocódigo de la clase Iterator, incluyendo las variables del objeto iterador y el método next().

11.9. Escriba el pseudocódigo para el borrado de entradas de una estructura asociativa extensible, incluyendo detalles del momento y forma de fusionar cajones. No se debe considerar la reducción del tamaño de la tabla de direcciones de cajones.

11.10. Sugiera una forma eficaz de comprobar si la tabla de direcciones de cajones en una asociación extensible se puede reducir en tamaño almacenando además un recuento en la tabla de direcciones de cajones. Escriba los detalles de cómo se debería mantener el recuento cuando se dividen, fusionan o borran los cajones. (*Nota*: la reducción del tamaño de la tabla de direcciones de cajones es una operación costosa y las inserciones subsecuentes pueden causar que la tabla vuelva a crecer. Por tanto, es mejor no reducir el tamaño tan pronto como se pueda, sino solamente si el número de entradas de índice es pequeño en comparación con el tamaño de la tabla de direcciones de los cajones).

11.11. Considere la relación *profesor* que se muestra en la Figura 11.1.

a. Construya un índice de mapa de bits sobre el atributo *sueldo*, dividiendo *sueldo* en cuatro intervalos: menos de 50.000 €, entre 50.000 € y menos de 60.000 €, entre 60.000 € y menos de 70.000 €, y 70.000 € o más.

b. Considere una consulta que solicite todos los profesores del departamento de Finanzas con un sueldo de 80.000 € o más. Describa los pasos para responder a la consulta y muestre los mapas de bits finales e intermedios construidos para responder la consulta.

11.12. ¿Cuál sería la ocupación de cada nodo hoja de un árbol B^+ si las entradas de índice se insertan en orden? Explique por qué.

11.13. Suponga que se tiene una relación r con n_r tuplas sobre la que se va a construir un índice secundario de árbol B^+.

a. Escriba una fórmula para el coste de construir el índice de árbol B^+ insertando un registro cada vez. Suponga que cada página contendrá de media f entradas, y que todos los niveles del árbol superiores a las hojas están en memoria.

b. Suponiendo un tiempo de acceso a disco de 10 milisegundos, ¿cuál es el coste de la construcción del índice para una relación con 10 millones de registros?

c. Sugiera el pseudocódigo para la construcción de abajo arriba del árbol B^+ descrito en la Sección 11.4.4. Puede suponer que se dispone de una función para ordenar grandes archivos de forma muy eficiente.

11.14. ¿Por qué podrían perder la secuencialidad los nodos hoja de una organización de archivo de árboles B^+?

a. Sugiera cómo se puede reorganizar la organización de archivo para recuperar la secuencialidad.

b. Una alternativa a la reorganización es asignar páginas hijas en unidades de n bloques, para algún n razonablemente grande. Cuando se asigna la primera hoja del árbol B, solo se usa un bloque de los bloques-n y el resto de páginas quedan libres. Si una página se divide, y su unidad de bloque-n está lleno, si asigna otra unidad de bloque-n y las primeras $n/2$ páginas hoja se sitúan en una unidad de bloque-n. Para simplificar, suponga que no se realizan operaciones de borrado.

i. ¿Cuál es la ocupación de espacio en el caso peor, suponiendo que no hay operaciones de borrado, después de que la primera unidad de bloque-n esté llena?

ii. ¿Puede ser que los nodos hoja asignados a una unidad de bloque-n no queden consecutivos, es decir, puede que dos nodos hoja se asignen a un bloque-n, pero otro nodo hoja entre los dos anteriores se asigne a un bloque-n diferente?

iii. Bajo la suposición razonable de que el espacio de la memoria intermedia es suficiente para almacenar un bloque-n, ¿cuántas búsquedas en disco se requieren para recorrer el nivel de hojas del árbol B^+ en el caso peor? Compare este número con el caso peor si a las páginas hoja se les asignan los bloques de uno en uno.

iv. La técnica de redistribuir los valores de los hermanos para mejorar la utilización del espacio probablemente sea más eficiente cuando se usa con el esquema de asignación anterior para bloques hoja. Explique por qué.

Ejercicios

11.15. ¿Cuándo es preferible utilizar un índice denso en vez de un índice disperso? Razone la respuesta.

11.16. ¿Cuál es la diferencia entre un índice con agrupación y un índice secundario?

11.17. Para cada árbol B^+ del Ejercicio práctico 11.3 indique los pasos involucrados en las siguientes consultas:

a. Encontrar los registros con un valor de la clave de búsqueda de 11.

b. Encontrar los registros con un valor de la clave de búsqueda entre 7 y 17, ambos inclusive.

11.18. La solución presentada en la Sección 11.3.4 para tratar las claves de búsqueda duplicadas añadía un atributo adicional a la clave de búsqueda. ¿Qué efecto tiene este cambio en la altura del árbol B^+?

11.19. Explique las diferencias entre la asociación abierta y la cerrada. Comente las ventajas de cada técnica en aplicaciones de bases de datos.

11.20. ¿Cuáles son las causas del desbordamiento de cajones en un archivo con una organización asociativa? ¿Qué se puede hacer para reducir la aparición del desbordamiento de cajones?

11.21. ¿Por qué una estructura asociativa no es la mejor elección para una clave de búsqueda en la que son probables las consultas de intervalos?

11.22. Suponga la relación $r(A, B, C)$, con un índice de árbol B^+ con la clave de búsqueda (A, B).

a. ¿Cuál es el coste en el peor caso posible de buscar registros que satisfagan $10 < A < 50$ al usar este índice en términos del número de registros recuperados n_1 y la altura h del árbol?

b. ¿Cuál es coste en el peor caso posible de buscar registros que satisfagan $10 < A < 50 \wedge 5 < B < 10$ al usar este índice en términos del número de registros n_2 que satisfacen la condición y de n_1 y de h, como se definieron antes?

c. ¿Bajo qué condiciones sobre n_1 y sobre n_2 el índice sería una forma eficiente de buscar los registros que satisfagan $10 < A < 50 \wedge 5 < B < 10$?

11.23. Suponga que hay que crear un índice de árbol B^+ sobre un gran número de nombres, en el que el tamaño máximo de un nombre puede ser grande (por ejemplo, 40 caracteres) y el tamaño medio también (digamos, 10 caracteres). Explique cómo se puede usar la compresión de prefijo para maximizar el grado de salida medio de los nodos internos.

11.24. Suponga una relación almacenada en una organización de archivo de árbol B^+. Suponga que los índices secundarios guardan identificadores de registro que son punteros a los registros del disco.

a. ¿Cuál sería el efecto sobre los índices secundarios si se produce una división de página en la organización del archivo?

b. ¿Cuál sería el coste de la actualización de todos los registros afectados en un índice secundario?

c. ¿Cómo soluciona este problema el uso de una clave de búsqueda como un identificador lógico de registro?

d. ¿Cuál es el coste adicional debido al uso de estos identificadores lógicos?

11.25. Indique la forma de calcular mapas de bits de existencia a partir de otros mapas de bits. Asegúrese de que la técnica funciona incluso con valores nulos, usando un mapa de bits para el valor *nulo*.

11.26. ¿Cómo afecta el cifrado de datos a los esquemas de índices? En particular, ¿cómo afectaría a los esquemas que intentan almacenar los datos de manera ordenada?

11.27. Nuestra descripción de la asociación estática asume que una gran extensión contigua de bloques de disco se puede ubicar en una tabla asociativa estática. Suponga que se pueden asignar solo C bloques contiguos. Sugiera cómo implementar la tabla asociativa si puede ser mucho más grande que los C bloques. El acceso al bloque debería seguir siendo eficiente.

Notas bibliográficas

En Cormen et ál. [1990] se pueden encontrar explicaciones acerca de las estructuras básicas utilizadas en la indexación y asociación. Los índices de árbol B se introdujeron por primera vez en Bayer [1972] y en Bayer y McCreight [1972]. Los árboles B^+ se tratan en Comer [1979], Bayer y Unterauer [1977] y Knuth [1973]. Las notas bibliográficas del Capítulo 15 proporcionan referencias a la investigación sobre los accesos concurrentes y las actualizaciones en los árboles B^+. Gray y Reuter [1993] proporcionan una buena descripción de los resultados en la implementación de árboles B^+.

Se han propuesto varias estructuras alternativas de árboles y basadas en árboles. Los **tries** son unos árboles cuya estructura está basada en los «dígitos» de las claves (por ejemplo, el índice de muescas de un diccionario, con una entrada para cada letra). Estos árboles podrían no estar equilibrados en el sentido que lo están los árboles B^+. Los *tries* se estudian en Ramesh et ál. [1989], Orenstein [1982], Litwin [1981] y Fredkin [1960]. Otros trabajos relacionados son los árboles B digitales de Lomet [1981].

Knuth [1973] analiza un gran número de técnicas de asociación distintas. Existen varias técnicas de asociación dinámica. Fagin et ál. [1979] introducen la asociación extensible. La asociación lineal se introduce en Litwin [1978] y Litwin [1980]; en Rathi et ál. [1990] se presentó una comparación de rendimiento con la asociación extensible. Una alternativa propuesta en Ramakrishna y Larson [1989] permite la recuperación en un solo acceso a disco al precio de una gran sobrecarga en una pequeña fracción de las modificaciones de la base de datos. La asociación dividida es una extensión de la asociación para varios atributos, y se trata en Rivest [1976], Burkhard [1976] y Burkhard [1979].

Vitter [2001] proporciona una extensa visión general de las estructuras de datos en memoria externa y algoritmos para ellas.

Los índices de mapas de bits y las variantes denominadas **índices por capas de bits** e **índices de proyección** se describen en O'Neil y Quass [1997]. Se introdujeron por primera vez en el gestor de archivos Model 204 de IBM sobre la plataforma AS 400. Proporcionan grandes ganancias de velocidad en ciertos tipos de consultas y actualmente se encuentran implementadas en la mayoría de los sistemas de bases de datos. Entre la investigación más reciente sobre índices de mapas de bits figura la de Wu y Buchmann [1998], Chan y Ioannidis [1998], Chan e Ioannidis [1999] y Johnson [1999].

Procesamiento de consultas

12

El **procesamiento de consultas** hace referencia a una serie de actividades implicadas en la extracción de datos de una base de datos. Estas actividades incluyen la traducción de consultas expresadas en lenguajes de alto nivel de bases de datos en expresiones implementadas en el nivel físico del sistema, así como transformaciones de optimización de consultas y la evaluación real de las mismas.

12.1. Descripción general

En la Figura 12.1 se muestran los pasos involucrados en el procesamiento de una consulta. Los pasos básicos son:

1. Análisis y traducción.
2. Optimización.
3. Evaluación.

Antes de empezar el procesamiento de una consulta, el sistema debe traducirla a una forma utilizable. Un lenguaje como SQL es adecuado para el uso humano, pero es poco apropiado para una representación interna en el sistema de la consulta. Una representación interna más útil estaría basada en el álgebra relacional extendido.

Por tanto, la primera acción que el sistema tiene que emprender para procesar una consulta es su traducción a su formato interno. Este proceso de traducción es similar al que realiza el analizador de un compilador. Durante la generación del formato interno de una consulta, el analizador comprueba la sintaxis de la consulta del usuario, verifica que los nombres de las relaciones que aparecen en ella sean nombres de relaciones que existen en la base de datos, etc. El sistema construye un árbol del análisis de la consulta, que se transformará en una expresión del álgebra relacional. Si la consulta estuviera expresada en términos de una vista, la fase de traducción también sustituye todas las referencias a vistas por las expresiones del álgebra relacional que las definen.[1] El análisis de lenguajes se describe en la mayoría de los libros de texto sobre compiladores.

Dada una consulta, normalmente hay distintos métodos para obtener la respuesta. Por ejemplo, ya se ha visto que en SQL se puede expresar una consulta de diferentes maneras. Cada consulta SQL puede traducirse a diversas expresiones del álgebra relacional. Además de esto, la representación de una consulta en el álgebra relacional especifica de manera parcial cómo evaluar la consulta; existen normalmente varias maneras de evaluar expresiones del álgebra relacional. Como ejemplo, considere la consulta:

select *sueldo*
from *profesor*
where *sueldo* < 75000;

Esta consulta se puede traducir en alguna de las siguientes expresiones del álgebra relacional:

- $\sigma_{sueldo\ <\ 75000}\ (\Pi_{sueldo}\ (profesor))$
- $\Pi_{sueldo}\ (\sigma_{sueldo\ <\ 75000}\ (profesor))$

Además, se puede ejecutar cada operación del álgebra relacional utilizando alguno de los diferentes algoritmos. Por ejemplo, para implementar la selección anterior se puede examinar cada tupla de *profesor* para encontrar las tuplas cuyo sueldo sea inferior a 75.000. Si se dispone de un índice de árbol B^+ sobre el atributo *sueldo*, se puede utilizar este índice para localizar las tuplas.

Para especificar completamente cómo evaluar una consulta, no basta con proporcionar la expresión del álgebra relacional, además hay que anotar en ella las instrucciones que especifiquen cómo evaluar cada operación. Estas anotaciones podrían indicar el algoritmo a usar para una operación específica o el índice o índices concretos a utilizar. Las operaciones del álgebra relacional anotadas con instrucciones sobre su evaluación reciben el nombre de **primitivas de evaluación.** La secuencia de operaciones primitivas que se pueden utilizar en la evaluación de una consulta establece un **plan de ejecución de la consulta** o un **plan de evaluación de la consulta.** En la Figura 12.2 se ilustra un plan de evaluación para nuestro ejemplo de consulta, en el que se especifica un índice concreto (denotado en la figura como «índice 1») para la operación selección. El **motor de ejecución de consultas** escoge un plan de evaluación, lo ejecuta y devuelve su respuesta a la consulta.

Los diferentes planes de evaluación de una consulta dada pueden tener costes distintos. No se puede esperar que los usuarios escriban las consultas de manera que sugieran el plan de evaluación más eficiente. En su lugar, es responsabilidad del sistema construir un plan de evaluación de la consulta que minimice el coste de la evaluación; esta tarea se denomina *optimización de consultas*. El Capítulo 13 describe en detalle la optimización de consultas.

Una vez elegido el plan de la consulta, esta se evalúa con el plan y se muestra su resultado.

La secuencia de pasos que se ha descrito para procesar una consulta es representativa; no todas las bases de datos la siguen exactamente. Por ejemplo, en lugar de utilizar la representación del álgebra relacional, varias bases de datos usan una representación del árbol de análisis con anotaciones basadas en la estructura de la consulta SQL. Sin embargo, los conceptos que se describen aquí constituyen la base del procesamiento de consultas en las bases de datos.

Para optimizar una consulta, el optimizador de consultas debe conocer el coste de cada operación. Aunque el coste exacto es difícil de calcular, dado que depende de muchos parámetros, como la memoria real disponible, es posible obtener una estimación aproximada del coste de ejecución para cada operación.

1 Para vistas materializadas, la expresión que define la vista ha sido ya evaluada y almacenada. Por tanto, se puede usar la relación almacenada en lugar de reemplazar los usos de la vista por la expresión que define la vista. Las vistas recursivas se tratan de manera diferente mediante un procedimiento de búsqueda de punto fijo, según se vio en la Sección 5.4 (consúltese también Apéndice C.3.6 en línea, en inglés).

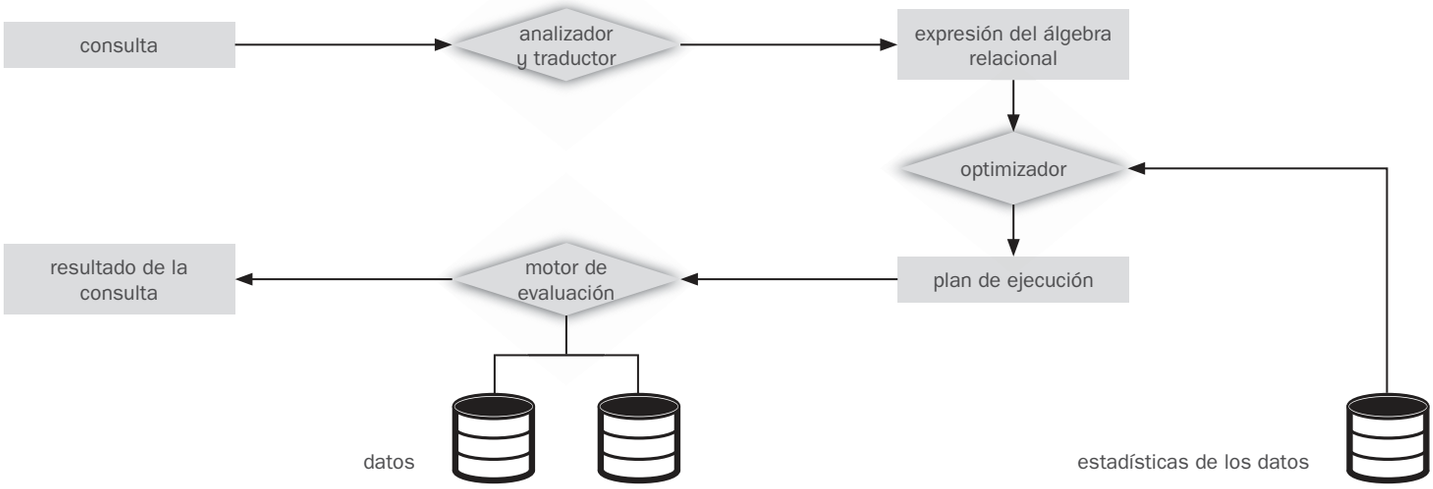


Figura 12.1. Pasos en el procesamiento de una consulta.

En este capítulo se estudia la forma de evaluar operaciones individuales en un plan de consulta y cómo estimar su coste; se vuelve a la optimización de consultas en el Capítulo 13.

En la Sección 12.2 se describe cómo se mide el coste de una consulta. Desde la Sección 12.3 hasta la 12.6 se estudia la evaluación de operaciones individuales del álgebra relacional. Varias operaciones se pueden agrupar en un **cauce,** en el que cada una de ellas empieza trabajando sobre sus tuplas de entrada del mismo modo que si fuesen generadas por otra operación.

En la Sección 12.7 se examina cómo coordinar la ejecución de varias operaciones de un plan de evaluación de consultas; en particular, cómo usar las operaciones encauzadas para evitar escribir resultados intermedios en disco.

Figura 12.2. Plan de ejecución de una consulta.

12.2. Medidas del coste de una consulta

Existen muchos posibles planes de evaluación de una consulta y es importante ser capaz de comparar las alternativas en términos de su coste (estimado) para elegir el mejor plan. Para ello hay que estimar el coste de las operaciones individuales y combinarlas para obtener el coste de un plan de evaluación de una consulta. Por tanto, más adelante en este capítulo se estudian los algoritmos de evaluación para cada operación viendo cómo estimar el coste de cada una.

El coste de la evaluación de una consulta se puede expresar en términos de diferentes recursos, incluyendo los accesos a disco, el tiempo que tarda la CPU en ejecutar una consulta y, en sistemas de bases de datos distribuidos o paralelos, el coste de la comunicación (que se estudiará más adelante en los Capítulos 18 y 19).

En grandes sistemas de bases de datos, el coste de acceso a los datos en disco es normalmente el más importante, ya que los accesos a disco son lentos comparados con las operaciones en memoria. Además, la velocidad de la CPU ha aumentado mucho más rápidamente que las velocidades de los discos. Por tanto, lo más probable es que el tiempo empleado en operaciones del disco siga dominando en el tiempo total de ejecución de una consulta. Las estimaciones del tiempo de CPU son más difíciles de establecer porque dependen de detalles de bajo nivel del código de ejecución. Aunque los optimizadores reales de consultas consideran los costes de CPU, aquí se ignoran y solo se considera el coste de los accesos a disco para medir el coste del plan de evaluación de una consulta.

Se usará el *número de transferencias de bloques de disco* y el *número de búsquedas de disco* para medir el coste de un plan de evaluación de una consulta. Si al subsistema de disco le lleva una media de t_T segundos transferir un bloque de datos, con un tiempo medio de acceso a bloque (tiempo de búsqueda en el disco más latencia rotacional) de t_S segundos, entonces una operación que transfiera b bloques y ejecute S búsquedas tardaría $b * t_T + S * t_S$ segundos. Los valores de t_T y de t_S se deben ajustar al disco usado, pero los valores normales de los discos de altas prestaciones actuales serían $t_S = 4$ milisegundos y $t_T = 0{,}1$ milisegundos, suponiendo un tamaño de bloque de 4 kilobytes y una velocidad de transferencia de 40 megabytes por segundo.[2]

Se puede refinar aún más la estimación del coste distinguiendo entre las lecturas y escrituras de bloques, dado que normalmente se tarda el doble de tiempo en escribir un bloque que en leerlo del disco (debido a que los sistemas de disco leen los sectores después de escribirlos para comprobar que la escritura fue correcta). Para simplificar se ignorará este detalle y se propone al lector que desarrolle estimaciones de coste más precisas para distintas operaciones.

Las estimaciones de coste que se proporcionan no incluyen el coste de escribir el resultado final de una operación en disco. Esto se tendrá en cuenta cuando sea preciso. Los costes de todos los algoritmos que se considera dependen del tamaño de la memoria intermedia en la memoria principal. En el mejor caso, todos los datos se pueden leer en las memorias intermedias y no es necesario acceder de nuevo al disco. En el peor caso se asume que la memoria intermedia puede contener solo unos pocos bloques de datos (aproximadamente un bloque por relación). Al presentar las estimaciones de coste se asume generalmente el peor caso.

2 Algunos sistemas de bases de datos ejecutan búsquedas y transferencias de bloque de prueba para estimar los costes medios de búsqueda y transferencia de bloque, como parte del proceso de instalación del software.

Además, aunque se asuma que los datos se deben leer inicialmente del disco, es posible que el bloque al que se acceda pueda estar en la memoria intermedia. De nuevo, para simplificar, se ignorará este efecto; como resultado el coste del acceso al disco real durante la ejecución de un plan puede ser menor que el coste estimado.

El **tiempo de respuesta** para un plan de evaluación de una consulta (es decir, el tiempo de reloj para ejecutar el plan), suponiendo que no se está realizando ninguna otra actividad en la computadora, tendría en cuenta todos los costes y se podría usar como una medida del coste del plan. Desafortunadamente, el tiempo de respuesta de un plan resulta muy difícil de estimar sin ejecutar realmente el plan, por las siguientes razones:

1. El tiempo de respuesta depende del contenido de la memoria intermedia cuando comienza la ejecución de la consulta; no se puede conocer esta información cuando se optimiza la consulta y resulta difícil tenerlo en cuenta aunque estuviese disponible.
2. En un sistema con varios discos, el tiempo de respuesta depende de cómo se distribuyan los accesos entre los discos, lo que resulta muy difícil de estimar sin una información detallada de la distribución de los datos.

Curiosamente, un plan puede tener un mejor tiempo de respuesta a costa de un consumo adicional de recursos. Por ejemplo, si un sistema tiene varios discos, un plan *A* que requiere lecturas adicionales de disco, pero realiza la lectura en paralelo entre todos los discos, puede acabar antes que otro plan *B* que tiene menos lecturas de disco, pero solo lee de uno. Sin embargo, si se ejecutan concurrentemente muchas consultas que usan el plan *A* el tiempo de respuesta total puede ser mayor que si se ejecutan las mismas consultas usando el plan *B*, ya que el plan *A* genera mayor carga en los discos.

El resultado es que en lugar de intentar minimizar el tiempo de respuesta, los optimizadores suelen intentar minimizar el **consumo de recursos** total de un plan de consulta. Nuestro modelo de estimación del tiempo total de acceso al disco (incluyendo la búsqueda y la transferencia de datos) es un ejemplo de este modelo basado en el consumo de recursos para el coste de una consulta.

12.3. Operación selección

En el procesamiento de consultas, el **explorador de archivo** es el operador de nivel más bajo para acceder a los datos. Los exploradores de archivo son algoritmos de búsqueda que localizan y recuperan los registros que cumplen una condición de selección. En los sistemas relacionales, el explorador de archivo permite leer una relación completa en aquellos casos en los que la relación se almacena en un único archivo dedicado.

12.3.1. Selecciones usando exploración de archivos e índices

Considere una operación de selección en una relación cuyas tuplas se almacenan juntas en un archivo. La forma más directa de realizar una selección es la siguiente:

- **A1 (búsqueda lineal).** En una búsqueda lineal se explora cada bloque del archivo y se comprueban todos los registros para determinar si satisfacen o no la condición de selección. Se necesita una búsqueda inicial para acceder al primer bloque del archivo. En el caso de que los bloques no estén almacenados de forma contigua, se requieren búsquedas adicionales, pero se ignora este efecto para simplificar.

  Aunque podría ser más lento que otros algoritmos para la implementación de la selección, el algoritmo de búsqueda lineal se puede aplicar a cualquier archivo, sin importar su ordenación, la presencia de índices o la naturaleza de la operación selección. El resto de algoritmos que se estudiarán no son aplicables en todos los casos, pero cuando lo son, son más rápidos que la búsqueda lineal.

  En la Figura 12.3 se muestra el coste estimado para una búsqueda lineal, así como para otros algoritmos. En la figura se utiliza h_i para representar la altura del árbol B^+. Los optimizadores reales normalmente presuponen que la raíz del árbol se encuentra en la memoria, ya que se accede a él con frecuencia. Algunos optimizadotes presuponen también que todos los nodos internos se encuentran en la memoria, ya que se accede a ellos con relativa frecuencia, y normalmente menos del 1 por ciento de los nodos de un árbol B^+ son nodos internos. La fórmula del coste se puede modificar de forma apropiada.

  Las estructuras de índices se denominan **caminos de acceso,** ya que proporcionan un camino mediante el cual acceder y localizar los datos. En el Capítulo 11 se indicaba que es eficiente leer los registros de un archivo en un orden que se corresponda aproximadamente con el orden físico. Recuerde que un *índice primario* (también llamado *índice de agrupación*) es un índice que permite que se puedan leer todos los registros de un archivo en un orden que se corresponde con el orden físico en el disco. Un índice que no es el índice primario se denomina *índice secundario*.

  Los algoritmos de búsqueda que usan un índice se denominan **exploración con índice.** Se usa un predicado de selección para guiarnos en la elección del índice para el procesamiento de la consulta. Los algoritmos de búsqueda que usan un índice son:

- **A2 (índice primario, igualdad basada en la clave).** Para una condición de igualdad en un atributo clave con un índice primario se puede utilizar el índice para recuperar el único registro que satisface la correspondiente condición de igualdad. La estimación del coste se muestra en la Figura 12.3.
- **A3 (índice primario, igualdad basada en un atributo no clave).** Se pueden recuperar varios registros mediante el uso de un índice primario cuando la condición de selección especifica una comparación de igualdad en un atributo *A* que no sea clave. La única diferencia con el caso anterior es que puede ser necesario recuperar varios registros. Sin embargo, estos registros estarían almacenados consecutivamente en el archivo, ya que el archivo se ordena según la clave de búsqueda. La estimación del coste se muestra en la Figura 12.3.
- **A4 (índice secundario, igualdad).** Las selecciones con una condición de igualdad pueden utilizar un índice secundario. Esta estrategia puede recuperar un único registro si la condición de igualdad es sobre una clave; puede que se recuperen varios registros si el campo índice no es clave.

  En el primer caso solo se obtiene un registro. El coste en tiempo es el mismo que para el de índice primario (A2).

  En el segundo caso, cada registro puede residir en un bloque diferente, que puede provocar una operación E/S por cada registro recuperado, y cada operación E/S requiere una búsqueda y una transferencia de bloque. El coste temporal en este caso es $(h_i + n) * (t_S + t_T)$, donde n es el número de registros recuperados, si cada registro se encuentra en un bloque de disco diferente, y las búsquedas de bloque están aleatorizadas. El coste en el peor caso podría ser peor que en la búsqueda lineal si se buscan un gran número de registros.

  Si la memoria intermedia es grande, puede que el registro que contenga el bloque ya se encuentre en ella. Se puede realizar una estimación del coste *esperado* o *promedio* de la selección teniendo en cuenta la probabilidad de que el bloque que contiene el registro ya se encuentre en la memoria intermedia. Para memorias intermedias grandes, la estimación será mucho menor que en el peor caso.

En ciertos algoritmos, incluyendo el A2, el uso de una organización de archivo en árbol B+ puede ahorrar un acceso, ya que los registros se almacenan en el nivel de las hojas del árbol.

Como se describió en la Sección 11.4.2, cuando los registros se almacenan en una organización de árbol B+ u otras organizaciones que puedan requerir la reubicación de los registros, los índices secundarios generalmente no almacenan punteros a los registros.[3] En su lugar, estos índices almacenan los valores de los atributos usados como la clave de búsqueda en una organización de árbol B+. El acceso a un registro mediante un índice secundario es por tanto más costoso: primero se busca en el índice secundario para encontrar los valores de la clave de búsqueda del índice primario y después se busca en el índice primario para encontrar los registros. La fórmula del coste descrita para los índices secundarios se tendrá que modificar adecuadamente si se usan estos índices.

	Algoritmo	Coste	Razón
A1	Búsqueda lineal	$t_S + b_r * t_T$	Una búsqueda inicial más b_r transferencias de bloques, siendo b_r el número de bloques en el archivo.
A1	Búsqueda lineal, igualdad en la clave	Caso medio $t_S + (b_r / 2) * t_T$	Como solo un registro, como mucho, satisface la condición, la búsqueda se termina en cuanto se encuentra el registro. En el peor caso se requiere transferir b_r bloques.
A2	Índice primario del árbol B+, igualdad en la clave	$(h_i + 1) * (t_T + t_S)$	(Donde h_i indica la altura del índice). La búsqueda del índice recorre la altura del árbol más una E/S para obtener el registro; cada operación de E/S requiere una búsqueda y una transferencia de bloque.
A3	Índice primario del árbol B+, igualdad en atributo no clave	$h_i * (t_T + t_S) + b * t_T$	Una búsqueda para cada nivel del árbol, una búsqueda para el primer bloque. Aquí b es el número de bloques que contienen registros con la clave de búsqueda indicada y se leen todos. Estos bloques son nodos hoja y se supone que son secuenciales (ya que es un índice primario), no requiriendo búsquedas adicionales.
A4	Índice secundario del árbol B+, igualdad en la clave	$(h_i + 1) * (t_T + t_S)$	Este caso es similar al del índice primario.
A4	Índice secundario del árbol B+, igualdad en atributo no clave	$(h_i + n) * (t_T + t_S)$	(Donde n es el número de registros obtenidos). El coste del recorrido es el mismo que para A3, pero cada registro puede estar en un bloque diferente, lo que requiere una búsqueda por registro. El coste puede ser muy alto si n es muy grande.
A5	Índice primario del árbol B+, comparación	$h_i * (t_T + t_S) + b * t_T$	Idéntico al caso A3, igualdad en atributo no clave.
A6	Índice secundario del árbol B+, comparación	$(h_i + n) * (t_T + t_S)$	Idéntico al caso A4, igualdad en atributo no clave.

Figura 12.3. Estimación del coste para los algoritmos de selección.

3 Recuerde que si se usan organizaciones de árboles B+ para almacenar relaciones, los registros se pueden trasladar entre bloques cuando los nodos hoja se dividen o fusionan, y cuando se redistribuyen los registros.

12.3.2. Selecciones con condiciones de comparación

Considere una selección de la forma $\sigma_{A \leq v}(r)$. Se pueden implementar utilizando una búsqueda lineal o con índices de alguna de las siguientes maneras:

- **A5 (índice primario, comparación).** Se puede utilizar un índice ordenado primario (por ejemplo, un índice primario de árbol B+) cuando la condición de selección sea una comparación. Para condiciones con comparaciones de la forma $A > v$ o $A \geq v$ se puede usar el índice primario sobre A para guiar la recuperación de las tuplas de la siguiente forma. Para el caso de $A \geq v$ se busca el valor de v en el índice para encontrar la primera tupla del archivo que tenga un valor de $A = v$. Un explorador de archivo que comience en esa tupla y llegue hasta el final del archivo devuelve todas las tuplas que satisfacen la condición. Para $A > v$, el explorador de archivo comienza con la primera tupla, tal que $A > v$. La estimación del coste en este caso es idéntica a la del caso A3.

  Para comparaciones de la forma $A < v$ o $A \leq v$ no es necesario buscar en el índice. Para el caso de $A < v$ se utiliza un simple explorador partiendo del inicio del archivo y continuando hasta (pero sin incluirlo) la primera tupla con el atributo $A = v$. El caso $A \leq v$ es similar, excepto que el explorador continúa hasta (pero sin incluir) la primera tupla con el atributo $A > v$. En ninguno de los dos casos el índice es de utilidad alguna.
- **A6 (índice secundario, comparación).** Se puede utilizar un índice secundario ordenado para guiar la recuperación bajo condiciones de comparación que contengan $<$, $\leq$, $\geq$, o $>$. Los bloques del índice del nivel más bajo se exploran, bien desde el valor más pequeño hasta v (para $<$ y $\leq$) o bien desde v hasta el valor máximo (para $>$ y $\geq$).

  El índice secundario proporciona punteros a los registros, pero para obtener los registros reales es necesario extraerlos usando los punteros. Este paso puede requerir una operación E/S por cada registro extraído, dado que los registros consecutivos pueden estar en diferentes bloques del disco; como antes, cada operación E/S requiere una búsqueda y una transferencia de bloque. Si el número de registros extraídos es grande, el uso del índice secundario puede ser incluso más caro que la búsqueda lineal. Por tanto, el índice secundario solo se debería usar si se seleccionan muy pocos registros.

12.3.3. Implementación de selecciones complejas

Hasta ahora solo se han considerado condiciones de selección simples de la forma $A\ op\ B$, donde op es una operación de igualdad o de comparación. Se revisarán a continuación predicados de selección más complejos.

- **Conjunción:** una *selección conjuntiva* es una selección de la forma:

$$\sigma_{\theta_1 \wedge \theta_2 \wedge \cdots \wedge \theta_n}(r)$$

- **Disyunción:** una *selección disyuntiva* es una selección de la forma:

$$\sigma_{\theta_1 \vee \theta_2 \vee \cdots \vee \theta_n}(r)$$

  Una condición disyuntiva se cumple mediante la unión de todos los registros que cumplen alguna de las condiciones θ_i.
- **Negación:** el resultado de una selección $\sigma_{\neg\theta}(r)$ es el conjunto de tuplas de r para las que la condición θ es falsa. En ausencia de valores nulos, este conjunto es simplemente el conjunto de tuplas que no están en $\sigma_\theta(r)$.

Se puede implementar una operación selección con una conjunción o una disyunción de condiciones sencillas utilizando alguno de los siguientes algoritmos:

- **A7 (selección conjuntiva utilizando un índice).** Inicialmente hay que determinar si para un atributo hay disponible algún camino de acceso en alguna de las condiciones simples. Si lo hay, cualquiera de los algoritmos de selección A2 hasta A6 puede recuperar los registros que cumplan esa condición. Se completa la operación mediante la comprobación en la memoria intermedia de que cada registro recuperado cumpla o no el resto de condiciones simples.

  Para reducir el coste, se elige un θ_i y uno de los algoritmos entre A1 y A6 para el que la combinación dé lugar al menor coste de $\sigma_{\theta_i}(r)$. El coste del algoritmo A7 está determinado por el coste del algoritmo elegido.
- **A8 (selección conjuntiva utilizando un índice compuesto).** Puede que se disponga de un *índice compuesto* (es decir, un índice sobre varios atributos) apropiado para algunas selecciones conjuntivas. Si la selección especifica una condición de igualdad en dos o más atributos y existe un índice compuesto en estos campos con atributos combinados, entonces se podría buscar en el índice directamente. El tipo de índice determina cuál de los algoritmos A2, A3 o A4 se utilizará.
- **A9 (selección conjuntiva mediante la intersección de identificadores).** Otra alternativa para implementar la operación selección conjuntiva implica la utilización de punteros a registros o identificadores de registros. Este algoritmo necesita índices con punteros a registros en los campos involucrados por cada condición individual. De este modo se explora cada índice en busca de punteros cuyas tuplas cumplan alguna condición individual. La intersección de todos los punteros recuperados forma el conjunto de punteros a tuplas que satisfacen la condición conjuntiva. Luego se usa el conjunto de punteros para recuperar los registros reales. Si no hubiera índices disponibles para algunas condiciones concretas, entonces habría que comprobar el resto de condiciones de los registros recuperados.

  El coste del algoritmo A9 es la suma de los costes de cada una de las exploraciones del índice más el coste de recuperar los registros en la intersección de las listas recuperadas de punteros. Este coste se puede reducir ordenando la lista de punteros y recuperando los registros en orden. Por tanto, (1) todos los punteros a los registros de un bloque van juntos, y así todos los registros seleccionados en el bloque se pueden recuperar usando una única operación E/S, y (2) los bloques se leen ordenados, minimizando el movimiento del brazo del disco. La Sección 12.4 describe algunos algoritmos de ordenación.
- **A10 (selección disyuntiva mediante la unión de identificadores).** Si se dispone de caminos de acceso en todas las condiciones de la selección disyuntiva, se explora cada índice en busca de punteros cuyas tuplas cumplan una condición individual. La unión de todos los punteros recuperados proporciona el conjunto de punteros a todas las tuplas que cumplen la condición disyuntiva. Después se utilizan estos punteros para recuperar los registros reales.

  Sin embargo, aunque solo una de las condiciones no tenga un camino de acceso, se tiene que realizar una búsqueda lineal en la relación para encontrar todas las tuplas que cumplan la condición. Por tanto, aunque solo exista una de estas condiciones en la disyunción, el método de acceso más eficiente es una exploración lineal, que comprueba durante la exploración la condición disyuntiva en cada tupla.

La implementación de selecciones con condiciones negativas se deja como ejercicio (Ejercicio práctico 12.6).

12.4. Ordenación

La ordenación de los datos juega un papel importante en los sistemas de bases de datos por dos razones. Primero, las consultas SQL pueden solicitar que los resultados queden ordenados. Segundo, e igualmente importante para el procesamiento de consultas, varias de las operaciones relacionales, como las reuniones, se pueden implementar eficientemente si las relaciones de entrada están ordenadas. Por este motivo se revisa la ordenación antes que la operación reunión en la Sección 12.5.

La ordenación se puede conseguir mediante la construcción de un índice en la clave de ordenación y utilizando luego ese índice para leer la relación de manera ordenada. No obstante, este proceso ordena la relación solo *lógicamente* a través de un índice, en lugar de *físicamente*. Por tanto, la lectura de las tuplas de manera ordenada podría implicar un acceso al disco (búsqueda más transferencia de bloque) para cada tupla, lo que puede ser muy costoso dado que el número de registros puede ser mucho mayor que el número de bloques. Por esta razón sería deseable ordenar las tuplas físicamente.

El problema de la ordenación se ha estudiado ampliamente para los casos en que la relación cabe completamente en la memoria principal, y para el caso en el que la relación es mayor que la memoria. En el primer caso se utilizan técnicas de ordenación clásicas como la ordenación rápida (*quick-sort*). Aquí se estudia cómo tratar el segundo caso.

12.4.1. Algoritmo ordenación por mezcla externa

La ordenación de relaciones que no caben en la memoria se denomina **ordenación externa.** La técnica más utilizada para la ordenación externa es normalmente el algoritmo de **ordenación por mezcla externa.** A continuación se describe el algoritmo de ordenación por mezcla externa. Sea M el número de bloques en la memoria intermedia de la memoria principal, es decir, el número de bloques de disco cuyos contenidos se pueden alojar en la memoria intermedia de la memoria principal.

1. En la primera etapa, se crean varias **secuencias** ordenadas; cada secuencia está ordenada, pero solo contiene parte de los registros de la relación.

   $i = 0$;
   repeat
   leer M bloques, bien de la relación o bien del resto de la relación, según cuál tenga menor número de bloques;
   ordenar la parte de la relación que está en la memoria;
   escribir los datos ordenados para archivo de secuencias R_i;
   $i = i + 1$;
   until el final de la relación
2. En la segunda etapa, las secuencias se *mezclan*. Supóngase que, por ahora, el número total de secuencias N es menor que M, así que se puede asignar un marco de página para cada secuencia y reservar espacio para guardar una página con el resultado. La etapa de mezcla se lleva a cabo de la siguiente manera:

   leer un bloque de cada uno de los N archivos R_i en un bloque de memoria intermedia de la memoria;
   repeat
   elegir la primera tupla (según el orden) de entre todas las páginas de la memoria intermedia;
   escribir la tupla y suprimirla de la página de la memoria intermedia;
   if la página de la memoria intermedia de alguna secuencia R_i está vacía **and not** fin-de-archivo(R_i)
   then leer el siguiente bloque de R_i en el bloque de la memoria intermedia;
   until todas las páginas de la memoria intermedia estén vacías

El resultado de la etapa de mezcla es la relación ya ordenada. El archivo de salida se almacena en una memoria intermedia para reducir el número de operaciones de escritura en el disco. La operación anterior de mezcla es una generalización de la mezcla de dos vías utilizada por el algoritmo normal de ordenación por mezcla en memoria; este mezcla N secuencias, por eso se llama **mezcla de N vías.**

En general, si la relación es mucho mayor que la memoria, se podrían generar M o más secuencias en la primera etapa y no sería posible asignar un bloque para cada secuencia durante la etapa de mezcla. En este caso, se realiza la operación de mezcla en varios ciclos. Como hay suficiente memoria para $M-1$ bloques en la memoria intermedia de entrada, cada mezcla puede tomar $M-1$ secuencias como entrada.

El *ciclo* inicial se realiza como sigue. Se mezclan las $M-1$ secuencias primeras (según se ha descrito ya en el punto 2) para obtener un única secuencia en el siguiente ciclo. Luego se mezclan de manera similar las siguientes $M-1$ secuencias, continuando así hasta que todas las secuencias iniciales se hayan procesado. En este punto, el número de secuencias se ha reducido por un factor de $M-1$. Si el número reducido de secuencias es todavía mayor o igual que M, se realiza otro ciclo con las secuencias creadas en el primer ciclo como entrada. Cada ciclo reduce el número de secuencias por un factor de $M-1$. Estos ciclos se repiten tantas veces como sea necesario, hasta que el número de secuencias sea menor que M; momento en el que un último ciclo genera el resultado ordenado.

En la Figura 12.4 se ilustran los pasos de la ordenación por mezcla externa en una relación ficticia. Por motivos didácticos suponga que solamente cabe una tupla en cada bloque ($f_r = 1$) y que la memoria puede contener como mucho tres marcos de página. Durante la etapa de mezcla se utilizan dos marcos de página como entrada y uno para la salida.

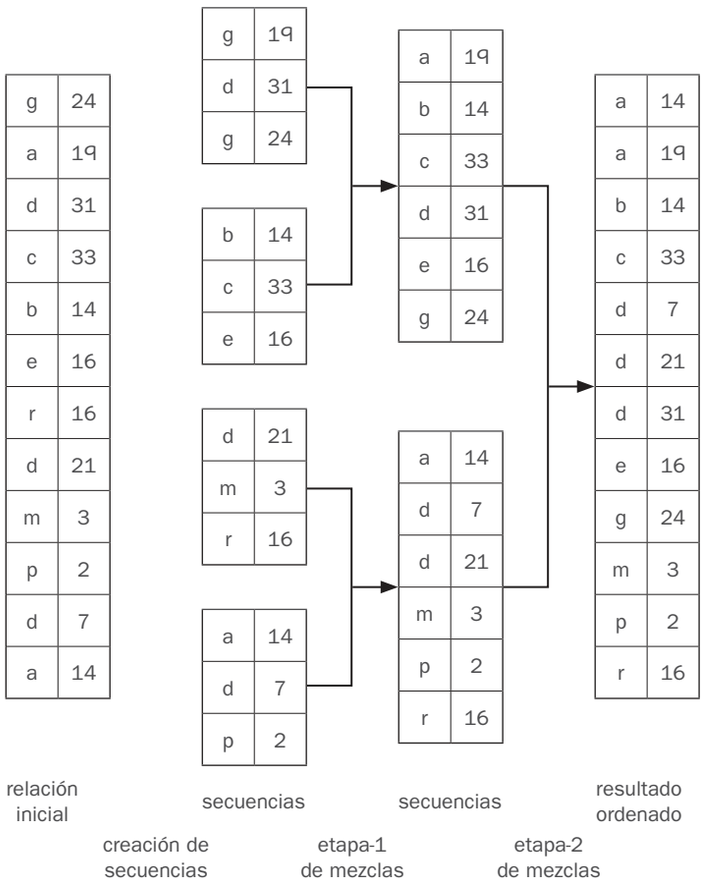


Figura 12.4. Ejemplo de ordenación por mezcla externa.

12.4.2. Análisis del coste de la ordenación por mezcla externa

El coste de acceso al disco de la ordenación por mezcla externa se calcula de la siguiente manera: sea b_r el número de bloques que contienen registros de la relación r. En la primera etapa, se lee y se copia de nuevo cada bloque de la relación, lo que da lugar a un total de $2b_r$ transferencias de bloques. El número inicial de secuencias es $\lceil b_r/M \rceil$. Puesto que el número de secuencias decrece en un factor de $M-1$ en cada ciclo de la mezcla, el número total de ciclos requeridos viene dado por la expresión $\lceil \log_{M-1}(b_r/M) \rceil$. Cada uno de estos ciclos lee y copia todos los bloques de la relación una vez, con dos excepciones. En primer lugar, el ciclo final puede producir la ordenación como resultado sin escribir el resultado en el disco. En segundo lugar, puede que haya secuencias que ni lean ni copien durante un ciclo (por ejemplo, si hay M secuencias que mezclar en un ciclo, se leen y se mezclan $M-1$, y no se accede a una de las secuencias durante ese ciclo). Si se ignora el ahorro (relativamente pequeño) debido a este último efecto, el número total de transferencias de bloques para la ordenación externa de la relación es:

$$b_r\,(2\lceil \log_{M-1}(b_r/M) \rceil + 1)$$

Aplicando esta ecuación al ejemplo de la Figura 12.4 se obtiene un total de $12 * (4 + 1) = 60$ transferencias de bloques, como se puede comprobar en la figura. Observe que este número no incluye el coste de escribir el resultado final.

También es necesario añadir el coste de la búsqueda en disco. La generación de ciclos requiere búsquedas para la lectura de datos de cada una de las secuencias, así como para escribir las secuencias. Durante la fase de mezcla, si se leen simultáneamente b_b bloques de cada secuencia (es decir, se asignan b_b bloques de memoria intermedia a cada secuencia), entonces cada paso de mezcla requeriría cerca de $\lceil b_r/b_b \rceil$ búsquedas para la lectura de los datos.[4] Aunque la salida se escriba secuencialmente, si está en el mismo disco que las secuencias de entrada, la cabeza se puede haber desplazado entre las escrituras de los bloques consecutivos; por tanto, habría que añadir un total de $2\lceil b_r/b_b \rceil$ búsquedas por cada paso de mezcla, excepto el paso final (ya que se asume que el resultado final no se escribe a disco). El número total de búsquedas es, por tanto:

$$2\lceil b_r/M \rceil + \lceil b_r/b_b \rceil (2\lceil \log_{M-1}(b_r/M) \rceil - 1)$$

Aplicando esta ecuación al ejemplo de la Figura 12.4 se obtiene un total de $8 + 12 * (2 * 2 - 1) = 44$ búsquedas si se establece el número de bloques por secuencia b_b como 1.

12.5. Operación reunión

En esta sección se estudian varios algoritmos para calcular la reunión de relaciones y para analizar sus costes asociados.

Se utiliza la palabra **equirreunión** para hacer referencia a las reuniones de la forma $r \bowtie_{r.A=s.B} s$, donde A y B son atributos o conjuntos de atributos de las relaciones r y s, respectivamente.

Utilizaremos como ejemplo la expresión:

$$estudiante \bowtie matricula$$

usando los mismos esquemas de relación que se usaron en el Capítulo 2. Suponga la siguiente información sobre las dos relaciones:

- Número de registros de *estudiante*: $n_{estudiante} = 5.000$.
- Número de bloques de *estudiante*: $b_{estudiante} = 100$.
- Número de registros de *matricula*: $n_{matricula} = 10.000$.
- Número de bloques de *matricula*: $b_{matricula} = 400$.

4 Para ser más precisos, dado que cada secuencia se lee por separado y se pueden obtener menos de b_b bloques al leer el final de una secuencia, es posible que se necesite una búsqueda adicional por cada secuencia. Para simplificar se ignorará este detalle.

12.5.1. Reunión en bucle anidado

En la Figura 12.5 se muestra un algoritmo sencillo para calcular la reunión zeta, $r \bowtie_\theta s$, de dos relaciones r y s. Este algoritmo se llama de **reunión en bucle anidado,** ya que básicamente consiste en un par de bucles **for** anidados. La relación r se denomina **relación externa** y s **relación interna** de la reunión, puesto que el bucle de r incluye al bucle de s. El algoritmo utiliza la notación $t_r \cdot t_s$, donde t_r y t_s son tuplas; $t_r \cdot t_s$ denota la tupla construida mediante la concatenación de los valores de los atributos de las tuplas t_r y t_s.

Al igual que el algoritmo de búsqueda lineal en archivos de la selección, el algoritmo de reunión en bucle anidado tampoco necesita índices y se puede utilizar sin importar la condición de la reunión. La manera de extender el algoritmo para calcular la reunión natural es directa, puesto que la reunión natural se puede expresar como una reunión zeta seguida de la eliminación de los atributos repetidos mediante una proyección. El único cambio que se necesita es un paso adicional para borrar los atributos repetidos de la tupla $t_r \cdot t_s$ antes de añadirla al resultado.

El algoritmo de reunión en bucle anidado es costoso, ya que examina cada pareja de tuplas en las dos relaciones. Considere el coste del algoritmo de reunión en bucle anidado. El número de pares de tuplas a considerar es $n_r * n_s$, donde n_r denota el número de tuplas en r, y n_s denota el número de tuplas en s. Para cada registro de r se tiene que realizar una exploración completa de s. En el peor caso, la memoria intermedia solamente puede contener un bloque de cada relación, necesitándose un total de $n_r * b_s + b_r$ transferencias de bloques, donde b_r y b_s denotan el número de bloques que contienen tuplas de r y de s, respectivamente. Solo se necesita una búsqueda por cada exploración de la relación interna s, dado que se lee secuencialmente, y un total de b_r búsquedas para leer r, por lo que resulta un total de $n_r + b_r$ búsquedas. En el mejor caso hay suficiente espacio como para que las dos relaciones quepan en la memoria, así que cada bloque se tendrá que leer solamente una vez; en consecuencia, solo se necesitarán $b_r + b_s$ transferencias de bloques, además de dos búsquedas.

```
for each tupla t_r in r do begin
   for each tupla t_s in s do begin
      comprobar si el par (t_r, t_s) satisface la condición θ de la reunión
      si la cumple, añadir t_r · t_s al resultado;
   end
end
```

Figura 12.5. Reunión en bucle anidado.

```
for each bloque B_r of r do begin
   for each bloque B_s of s do begin
      for each tupla t_r in B_r do begin
         for each tupla t_s in B_s do begin
            comprobar si el par (t_r, t_s) satisface la condición θ de la reunión
            si la cumple, añadir t_r · t_s al resultado;
         end
      end
   end
end
```

Figura 12.6. Reunión en bucle anidado por bloques.

Si una de las relaciones cabe en la memoria por completo, es útil usar esa relación como la relación más interna, ya que solamente será necesario leer una vez la relación del bucle más interno. Por lo tanto, si s es lo suficientemente pequeña para caber en la memoria principal, esta estrategia necesita solamente un total de $b_r + b_s$ transferencias de bloques y dos búsquedas (el mismo coste que en el caso en que las dos relaciones quepan en la memoria).

Considere ahora el caso de la reunión natural de *estudiante* y *matricula*. Suponga que, por el momento, no hay ningún índice en ninguna de las relaciones y que no se desea crear ninguno. Se pueden utilizar los bucles anidados para calcular la reunión; suponga que *estudiante* es la relación externa y que *matricula* es la relación interna de la reunión. Se tendrán que examinar $5.000 * 10.000 = 50 * 10^6$ pares de tuplas. En el peor caso el número de transferencias de bloques será de $5.000 * 400 + 100 = 2.000.100$, más $5.000 + 100 = 5.100$ búsquedas. En el mejor escenario, sin embargo, se tienen que leer ambas relaciones solamente una vez y realizar el cálculo. Este cálculo necesita a lo sumo $100 + 400 = 500$ transferencias de bloques más dos búsquedas (una mejora significativa respecto de la peor situación posible). Si se hubiera utilizado *matricula* como la relación del bucle externo y *estudiante* para el bucle interno, el coste en el peor caso de la última estrategia habría sido menor: $10.000 * 100 + 400 = 1.000.400$ transferencias de bloques más 10.400 búsquedas. El número de transferencias de bloques es significativamente menor y, aunque el número de búsquedas es superior, el coste total se reduce, suponiendo que $t_S = 4$ milisegundos y que $t_T = 0,1$ milisegundos.

12.5.2. Reunión en bucle anidado por bloques

Si la memoria intermedia es demasiado pequeña para contener las dos relaciones por completo, todavía se puede lograr un mayor ahorro en los accesos a los bloques si se procesan las relaciones por bloques en lugar de por tuplas. En la Figura 12.6 se muestra la **reunión en bucle anidado por bloques,** que es una variante de la reunión en bucle anidado en la que se empareja cada bloque de la relación interna con cada bloque de la relación externa. En cada par de bloques se empareja cada tupla de un bloque con cada tupla del otro bloque para generar todos los pares de tuplas. Al igual que antes, se añaden al resultado todas las parejas de tuplas que satisfacen la condición de la reunión.

La diferencia principal en coste entre la reunión en bucle anidado por bloques y la reunión en bucle anidado básica es que, en el peor caso, cada bloque de la relación interna s se lee solamente una vez por cada *bloque* de la relación externa, en lugar de una vez por cada *tupla* de la relación externa. Por tanto, en el peor caso, habrá un total de $b_r * b_s + b_r$ transferencias de bloques, donde b_r y b_s indican respectivamente el número de bloques que contienen registros de r y de s, respectivamente. Cada exploración de la relación interna requiere una búsqueda y la exploración de la relación externa una búsqueda por bloque, dando un total de $2 * b_r$ búsquedas. Evidentemente, será más eficiente utilizar la relación más pequeña como la relación externa, en caso de que ninguna de ellas quepa en la memoria. En el mejor caso, cuando la relación interna quepa en la memoria, habrá que realizar $b_r + b_s$ transferencias de bloques y solo dos búsquedas (en este caso se escogería la relación de menor tamaño como relación interna).

En el ejemplo del cálculo de *estudiante* $\bowtie$ *matricula* se usará ahora el algoritmo de reunión en bucle anidado por bloques. En el peor caso hay que leer una vez cada bloque de *matricula* por cada bloque de *estudiante*. Así, en el peor caso son necesarias un total de $100 * 400 + 100 = 40.100$ transferencias de bloques más $2 * 100 = 200$ búsquedas. Este coste supone una mejora importante respecto a las $5.000 * 400 + 100 = 2.000.100$ transferencias de bloques más 5.100 búsquedas necesarias en el peor caso para la reunión en bucle anidado básica. El coste en el mejor caso sigue siendo el mismo, es decir, $100 + 400 = 500$ transferencias de bloques y dos búsquedas.

El rendimiento de los procedimientos de bucle anidado y bucle anidado por bloques se puede mejorar aún más:

- Si los atributos de la reunión en una reunión natural o en una equirreunión forman una clave de la relación interna, entonces el bucle interno puede finalizar tan pronto como se encuentre la primera correspondencia.
- En el algoritmo en bucle anidado por bloques, en lugar de utilizar bloques de disco como la unidad de carga de bloques de la relación externa, se puede utilizar el mayor tamaño que quepa en la memoria, mientras quede suficiente espacio para las memorias intermedias de la relación interna y la salida. En otras palabras, si la memoria tiene M bloques, se leen $M-2$ bloques de la relación externa de una vez, y cuando se lee cada bloque de la relación interna se reúne con los $M-2$ bloques de la relación externa. Este cambio reduce el número de exploraciones de la relación interna de b_r a $\lceil b_r/(M-2) \rceil$, donde b_r es el número de bloques de la relación externa. El coste total es $\lceil b_r/(M-2) \rceil * b_s + b_r$ transferencias de bloques y $2\lceil b_r/(M-2) \rceil$ búsquedas.
- Se puede explorar el bucle interno alternativamente hacia adelante y hacia atrás. Este método de búsqueda ordena las peticiones de bloques de disco de tal manera que los datos restantes en la memoria intermedia de la búsqueda anterior se reutilizan, reduciendo de este modo el número de accesos a disco necesarios.
- Si se dispone de un índice en un atributo de la reunión del bucle interno se pueden sustituir las búsquedas en archivos por búsquedas más eficientes en el índice. Esta optimización se describe en la Sección 12.5.3.

12.5.3. Reunión en bucle anidado indexada

En una reunión en bucle anidado (Figura 12.5), si se dispone de un índice sobre el atributo de la reunión del bucle interno, se pueden sustituir las exploraciones de archivo por búsquedas en el índice. Para cada tupla t_r de la relación externa r, se utiliza el índice para buscar tuplas en s que cumplan la condición de reunión con la tupla t_r.

Este método de reunión se denomina **reunión en bucle anidado indexada;** se puede utilizar cuando existen índices y cuando se crean índices temporales con el único propósito de evaluar la reunión.

La búsqueda de tuplas en s que cumplan las condiciones de la reunión con una tupla dada t_r es esencialmente una selección en s. Por ejemplo, considere *estudiante* $\bowtie$ *matricula*. Suponga que se tiene una tupla de *estudiante* con *ID* «00128». Entonces, las tuplas relevantes de s son aquellas que satisfacen la selección «$ID = 00128$».

El coste de la reunión en bucle anidado indexada se puede calcular como se indica a continuación. Para cada tupla en la relación externa r se realiza una búsqueda en el índice para s recuperando las tuplas apropiadas. En el peor caso solo hay espacio en la memoria intermedia para una página de r y una página del índice. Por tanto, son necesarias b_r operaciones de E/S para leer la relación r, donde b_r denota el número de bloques que contienen registros de r; cada E/S requiere una búsqueda y una transferencia de bloque, ya que la cabeza del disco se puede haber trasladado entre cada E/S. Para cada tupla de r se realiza una búsqueda en el índice para s. Por tanto, el coste temporal de la reunión se puede calcular como $b_r\,(t_T + t_S) + n_r * c$, donde n_r es el número de registros de la relación r, y c es el coste de una única selección en s utilizando la condición de la reunión. Ya se vio en la Sección 12.3 cómo estimar el coste del algoritmo de una única selección (posiblemente utilizando índices) cuyo cálculo proporciona el valor de c.

La fórmula del coste indica que, si hay índices disponibles en ambas relaciones r y s, normalmente es más eficiente usar como relación más externa aquella que tenga menos tuplas.

Por ejemplo, considere una reunión en bucle anidado indexada de *estudiante* $\bowtie$ *matricula*, con *estudiante* como relación externa. Suponga también que *matricula* tiene un índice de árbol B$^+$ primario en el atributo de la reunión *ID*, que contiene 20 entradas en promedio por cada nodo del índice. Dado que *matricula* tiene 10.000 tuplas, la altura del árbol es 4, y será necesario un acceso más para encontrar el dato real. Como $n_{estudiante}$ es 5.000, el coste total es $100 + 5000 * 5 = 25.100$ accesos a disco, cada uno de los cuales requiere una búsqueda y una transferencia de bloque. En cambio, como se indicó anteriormente, se necesitan 40.100 transferencias de bloque más 200 búsquedas para una reunión en bucle anidado por bloques. Aunque se ha reducido el número de transferencias de bloques, el coste de la búsqueda se ha incrementado, aumentando el coste total, dado que una búsqueda es considerablemente más costosa que una transferencia de bloque. Sin embargo, si se tuviese una selección sobre la relación *estudiante* que redujera significativamente el número de filas, la reunión en bucle anidado sería significativamente más rápida que una reunión en bucle anidado por bloques.

12.5.4. Reunión por mezcla

El algoritmo de **reunión por mezcla** (también llamado algoritmo de **reunión por ordenación-mezcla**) se puede utilizar para calcular reuniones naturales y equirreuniones. Sean $r(R)$ y $s(S)$ las relaciones que vamos a utilizar para realizar la reunión natural, y sean $R \cap S$ sus atributos en común. Suponga que ambas relaciones están ordenadas según los atributos de $R \cap S$. Por tanto, su reunión se puede calcular mediante un proceso muy parecido a la etapa de mezcla del algoritmo de ordenación por mezcla.

12.5.4.1. Algoritmo de reunión por mezcla

El algoritmo de reunión por mezcla se muestra en la Figura 12.7. En este algoritmo, *AtribsReunión* se refiere a los atributos de $R \cap S$ y, a su vez, $t_r \bowtie t_s$, donde t_r y t_s son las tuplas que tienen los mismos valores en *AtribsReunión*, y denota la concatenación de los atributos de las tuplas, seguida de una proyección para no incluir los atributos repetidos. El algoritmo de reunión por mezcla asocia un puntero con cada relación. Al comienzo, estos punteros apuntan a la primera tupla de sus respectivas relaciones. Según avanza el algoritmo, el puntero se mueve a través de la relación. De este modo se leen en S_s un grupo de tuplas de una relación con el mismo valor en los atributos de la reunión. El algoritmo de la Figura 12.7 *necesita* que cada conjunto de tuplas S_s quepa en la memoria principal; más adelante, en este apartado, se examinarán extensiones del algoritmo que evitan este supuesto. Después, las tuplas correspondientes de la otra relación (si las hay) se leen y se procesan según se están leyendo.

En la Figura 12.8 se muestran dos relaciones que están ordenadas en su atributo de la reunión $a1$. Es instructivo examinar los pasos del algoritmo de reunión por mezcla con las relaciones que se muestran en la figura.

El algoritmo de reunión por mezcla de la Figura 12.7 requiere que cada uno de los conjuntos S_s de todas las tuplas con el mismo valor para los atributos de la reunión quepan en la memoria principal. Este requisito suele cumplirse, incluso aunque la relación s sea grande. Si existen algunos atributos de reunión para los que S_s es mayor que la memoria disponible, se puede realizar una reunión en bucle anidado para esos conjuntos S_s, casándolos con los correspondientes bloques de tuplas en r con los mismos valores para los atributos de reunión.

Si cualquiera de las relaciones de entrada r o s no están ordenadas en los atributos de la reunión, se pueden ordenar primero, y después usar el algoritmo de reunión por unión. Este algoritmo también se puede extender fácilmente para realizar la equirreunión.

```
pr:= dirección de la primera tupla de r;
ps:= dirección de la primera tupla de s;
while (ps ≠ null and pr ≠ null) do
  begin
    t_s:= tupla a la que apunta ps;
    S_s:= {t_s};
    hacer que ps apunte a la siguiente tupla de s;
    done := false;
    while (not done and ps ≠ null) do
      begin
        t_s': = tupla a la que apunta ps;
        if (t_s' [AtribsReunión] = t_s [AtribsReunión])
          then begin
            S_s:= S_s ∪ {t_s'};
            hacer que ps apunte a la siguiente tupla de s;
            end
          else done := true;
      end
    t_r:= tupla a la que apunta pr;
    while (pr ≠ null and t_r [AtribsReunión] < t_s [AtribsReunión]) do
      begin
        hacer que pr apunte a la siguiente tupla de r;
        t_r:= tupla a la que apunta pr;
      end
    while (pr ≠ null and t_r [AtribsReunión] = t_s [AtribsReunión]) do
      begin
        for each t_s in S_s do
          begin
            añadir t_s ⋈ t_r al resultado;
          end
        hacer que pr apunte a la siguiente tupla de r;
        t_r:= tupla a la que apunta pr;
      end
    end
```

Figura 12.7. Reunión por mezcla.

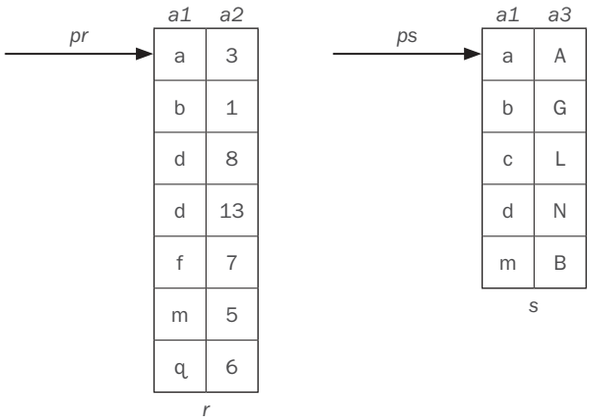


Figura 12.8. Relaciones ordenadas para la reunión por mezcla.

12.5.4.2. Análisis de coste

Una vez las relaciones están ordenadas, las tuplas con el mismo valor en los atributos de la reunión aparecerán consecutivamente. De este modo solamente es necesario leer ordenadamente cada tupla una vez y, como resultado, leer también cada bloque solamente una vez. Puesto que solo se recorre en una pasada ambos archivos (suponiendo que todos los conjuntos S_s caben en la memoria), el método de reunión por mezcla resulta eficiente; el número de transferencias de bloques es igual a la suma de los bloques en los dos archivos, $b_r + b_s$.

Suponiendo que se asignen b_b bloques de memoria intermedia para cada relación, el número de búsquedas requeridas sería $\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil$. Puesto que las búsquedas son mucho más costosas que las transferencias de bloques, tiene sentido asignar varios bloques de la memoria intermedia para cada relación, siempre que haya memoria disponible. Por ejemplo, con $t_T = 0{,}1$ milisegundos por bloque de 4 kilobytes y $t_S = 4$ milisegundos, el tamaño de la memoria intermedia es de 400 bloques (o 1,6 megabytes), por lo que el tiempo de búsqueda sería de 4 milisegundos por cada 40 milisegundos de tiempo de transferencia; en otras palabras, el tiempo de búsqueda sería solo el 10 por ciento del tiempo de transferencia.

Si alguna de las relaciones de entrada r o s no está ordenada según los atributos de la reunión, hay que ordenarlas primero; debemos añadir el coste de las ordenaciones al coste anterior. Si algunos de los conjuntos S_s no caben en la memoria, el coste se incrementa ligeramente.

Suponga que se aplica el esquema de reunión por mezcla al ejemplo de *estudiante* $\bowtie$ *matricula*. En este caso, el atributo de la reunión es *ID*. Suponga que las dos relaciones se encuentran ya ordenadas según el atributo de la reunión *ID*. En este caso, la reunión por mezcla emplea un total de $400 + 100 = 500$ transferencias de bloques. Si se asume que en el peor caso solo se asigna un bloque de memoria intermedia para cada relación de entrada (es decir, $b_b = 1$), se necesitarían un total de $400 + 100 = 500$ búsquedas; en realidad se puede hacer que b_b sea mucho mayor ya que solo se necesitan bloques de la memoria intermedia para dos relaciones, y el coste de búsqueda sería significativamente menor.

Suponga que las relaciones no están ordenadas y que el tamaño de la memoria en el peor caso es de solo tres bloques. El coste en este caso se calcularía:

1. Usando la fórmula desarrollada en la Sección 12.4, la ordenación de la relación *matricula*, hacen falta $\lceil \log_{3-1}(400/3) \rceil = 8$ pasos de mezcla. La relación de esta relación necesitaría $400 * (2\lceil \log_{3-1}(400/3) \rceil + 1)$, o 6.800 transferencias de bloques, con otras 400 transferencias para escribir el resultado. El número de búsquedas necesarias es $2 * \lceil 400/3 \rceil + 400 * (2 * 8 - 1)$, 6.268 búsquedas para ordenar y 400 búsquedas para escribir el resultado, con un total de 6.668 búsquedas dado que solo hay disponible un bloque de memoria intermedia para cada ciclo.
2. Análogamente, la ordenación de *impositor* necesita $\lceil \log_{3-1}(100/3) \rceil = 6$ pasos de mezcla y $100 * (2\lceil \log_{3-1}(100/3) \rceil + 1)$, 1.300 transferencias de bloques, con otras 100 transferencias para escribir el resultado. El número de búsquedas necesarias es $2 * \lceil 100/3 \rceil + 100 * (2 * 6 - 1) = 1.164$ búsquedas para ordenar, y 100 búsquedas para escribir el resultado, con un total de 1.264 búsquedas.
3. Finalmente, la mezcla de ambas relaciones necesita $400 + 100 = 500$ transferencias de bloques y 500 búsquedas.

Por tanto, el coste total es de 9.100 transferencias de bloques más 8.932 búsquedas si las relaciones no están ordenadas y el tamaño de memoria es solo de 3 bloques.

Con un tamaño de memoria de 25 bloques y sin tener ordenadas las relaciones, el coste de la ordenación seguida de la reunión por mezcla sería:

1. La ordenación de *matricula* se puede hacer en un único paso de mezcla y requiere un total de $400 * (2\lceil \log_{24}(400/25)\rceil + 1) = 1.200$ transferencias de bloques. De forma similar, la ordenación de *estudiante* emplea 300 transferencias de bloques. Escribir la salida ordenada a disco requiere $400 + 100 = 500$ transferencias de bloques, y el paso de mezcla necesita 500 transferencias de bloques para volver a leer los datos. Al sumar estos costes se calcula un total de 2.500 transferencias de bloques.
2. Si se asume que solo se asigna un bloque de memoria intermedia para cada secuencia, el número de búsquedas requeridas en este caso es $2 * \lceil 400/25\rceil + 400 + 400 = 832$ búsquedas para ordenar *matricula* y escribir la salida ordenada a disco y, análogamente, $2 * \lceil 100/25\rceil + 100 + 100 = 208$ para *estudiante*, más $400 + 100 = 500$ búsquedas para la lectura de los datos ordenados en el paso de reunión por mezcla. Al sumar estos costes se calcula un total de 1.640 búsquedas.

   Se puede reducir significativamente el número de búsquedas asignando más bloques de memoria intermedia en cada secuencia. Por ejemplo, si se asignan cinco bloques de memoria intermedia en cada secuencia y para la salida de la mezcla de cuatro secuencias de *estudiante*, el coste se reduce de 208 a $2 * \lceil 100/25\rceil + \lceil 100/5\rceil + \lceil 100/5\rceil = 48$ búsquedas. Si en el paso de reunión por mezcla se asignan 12 bloques para ambas relaciones *matricula* y *estudiante*, el número de búsquedas para este paso se reduce de 500 a $\lceil 400/12\rceil + \lceil 100/12\rceil = 43$ búsquedas. Por tanto, el número total de búsquedas es 251.

Por tanto, el coste total es de 2.500 transferencias de bloques más 251 búsquedas si las relaciones no están ordenadas y el tamaño de memoria es de 25 bloques.

12.5.4.3. Reunión por mezcla híbrida

Se puede realizar una variación de la operación de reunión por mezcla sobre tuplas no ordenadas, si existen índices secundarios en los atributos de reunión. El algoritmo explora los registros mediante los índices, lo que hace que se obtengan en el orden de la ordenación. Esta variación presenta una desventaja significativa, ya que los registros pueden encontrarse divididos por todos los bloques de archivo. Por tanto, los accesos a las tuplas pueden generar accesos a bloques de disco, lo que resulta costoso.

Para evitar este coste se puede usar una técnica de reunión por mezcla híbrida que combina los índices con la reunión por mezcla. Suponga que una de las relaciones está ordenada, que la otra está desordenada pero tiene un índice de árbol B^+ sobre uno de los atributos de la reunión. El **algoritmo de reunión por mezcla híbrida** mezcla la relación ordenada con las entradas hoja del índice secundario del árbol B^+. El archivo resultante contiene las tuplas de la relación ordenada y las direcciones de las tuplas de la relación sin ordenar. El archivo resultado se ordena por las direcciones de las tuplas de la relación sin ordenar, lo que permite un acceso eficiente a las tuplas correspondientes, en el orden físico de almacenamiento, para completar la reunión. Se deja como ejercicio la extensión de esta técnica para manejar dos relaciones sin ordenar.

12.5.5. Reunión por asociación

Al igual que en el algoritmo de reunión por mezcla, se puede utilizar el algoritmo de reunión por asociación para implementar reuniones naturales y equirreuniones. En el algoritmo de reunión por asociación se utiliza una función de asociación h para dividir las tuplas de ambas relaciones. La idea fundamental es dividir las tuplas de cada relación en conjuntos con el mismo valor de la función de asociación en los atributos de la reunión.

Se da por supuesto que:

- h es una función de asociación que asigna a los *AtribsReunión* los valores $\{0, 1, \ldots, n_h\}$, donde los *AtribsReunión* denotan los atributos comunes de r y s utilizados en la reunión natural.
- $r_0, r_1 \ldots, r_{n_h}$ denotan las particiones de las tuplas de r, inicialmente todas vacías. Cada tupla $t_r \in r$ se pone en la partición r_i, donde $i = h(t_r\,[AtribsReunión])$.
- $s_0, s_1 \ldots, s_{n_h}$ denotan las particiones de las tuplas de s, inicialmente todas vacías. Todas las tuplas $t_s \in s$ se ponen en la partición s_i, donde $i = h(t_s[AtribsReunión])$.

La función de asociación h debería tener las «buenas» propiedades de aleatoriedad y uniformidad que se estudiaron en el Capítulo 11. La división de las relaciones se muestra de manera gráfica en la Figura 12.9.

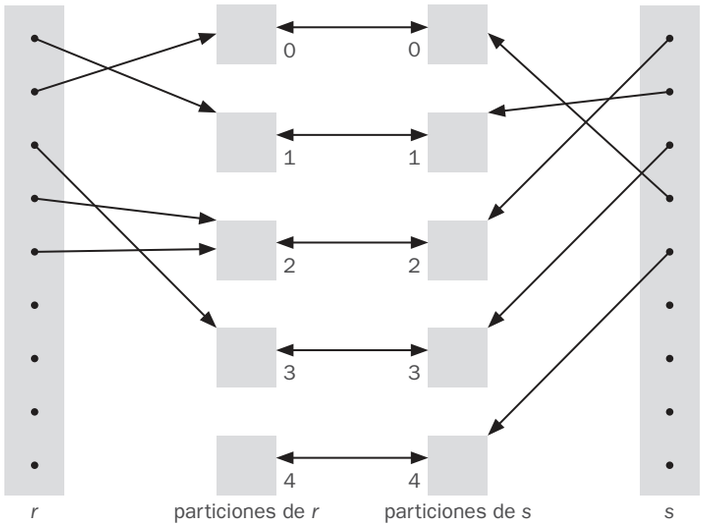


Figura 12.9. División por asociación de relaciones.

12.5.5.1. Fundamentos

La idea en la que se basa el algoritmo de reunión por asociación es la siguiente. Suponga que una tupla de r y una tupla de s satisfacen la condición de la reunión; por tanto, tendrán el mismo valor en los atributos de la reunión. Si el valor se asocia con algún valor i, la tupla de r tiene que estar en r_i y la tupla de s en s_i. De este modo solamente es necesario comparar las tuplas de r en r_i con las tuplas de s en s_i; no es necesario compararlas con las tuplas de s de ninguna otra partición.

Por ejemplo, si d es una tupla de *estudiante*, c una tupla de *matricula* y h una función de asociación en los atributos *ID* de las tuplas, solamente hay que comprobar d y c si $h(c) = h(d)$. Si $h(c) \neq h(d)$, entonces c y d poseen valores distintos de *ID*. Sin embargo, si $h(c) = h(d)$ hay que comprobar que c y d tengan los mismos valores en los atributos de la reunión, ya que es posible que c y d tengan valores diferentes de *ID*, con el mismo valor de la función de asociación.

En la Figura 12.10 se muestran los detalles del algoritmo de **reunión por asociación** para calcular la reunión natural de las relaciones r y s. Como en el algoritmo de reunión por mezcla, $t_r \bowtie t_s$ denota la concatenación de los atributos de las tuplas de t_r y t_s, seguida de la proyección para eliminar los atributos repetidos. Después de la división de las relaciones, el resto del código de reunión por asociación realiza una reunión en bucle anidado indexada separada en cada uno de los pares de partición i, siendo $i = 0, \ldots, n_h$. Para lograr esto, primero se **construye** un índice asociativo en cada s_i y luego se **prueba** (es decir, se busca en s_i) con las tuplas de r_i. La relación s es la **entrada de construcción** y r es la **entrada de prueba.**

El índice asociativo en s_i se crea en la memoria, así que no es necesario acceder al disco para recuperar las tuplas. La función de asociación utilizada para construir este índice asociativo debe ser distinta de la función de asociación h utilizada anteriormente, pero aún se aplica exclusivamente a los atributos de la reunión. A lo largo de la reunión en bucle anidado indexada, el sistema utiliza este índice asociativo para recuperar los registros que concuerden con los registros de la entrada de prueba.

```
/* División de s */
for each tupla t_s in s do begin
   i:= h(t_s [AtribsReunión]);
   H_si:= H_si ∪ {t_s};
end
/* División de r */
for each tupla t_r in r do begin
   i:= h(t_r [AtribsReunión]);
   H_ri:= H_ri ∪ {t_r};
end
/* Realizar la reunión de cada partición */
for i:= 0 to n_h do begin
   leer H_si y construir un índice asociativo en su memoria;
   for each tupla t_r in H_ri do begin
      explorar el índice asociativo en H_si para localizar todas las tuplas t_s
         tales que t_s [AtribsReunión] = t_r [AtribsReunión];
      for each tupla t_s que concuerde in H_si do begin
         añadir t_r ⋈ t_s al resultado;
      end
   end
end
```

Figura 12.10. Reunión por asociación.

Las etapas de construcción y prueba solo necesitan un único paso a través de las entradas de construcción y prueba. La extensión del algoritmo de reunión por asociación para calcular equirreuniones más generales es directa.

Se tiene que elegir un valor de n_h lo bastante grande como para que, para cada i, las tuplas de la partición s_i de la relación de construcción, junto con el índice asociativo de la partición, quepan en la memoria. Pero no tienen por qué caber en la memoria las particiones de la relación que se prueban. Sería obviamente mejor utilizar la relación de entrada más pequeña como la relación de construcción. Si el tamaño de la relación de construcción es de b_s bloques, entonces para que cada una de las n_h particiones tenga un tamaño menor o igual que M, n_h debe ser al menos $\lceil b_s/M \rceil$. Con más exactitud, hay que tener en cuenta también el espacio adicional ocupado por el índice asociativo de la partición, incrementando n_h según corresponda. Por simplificar, en estos análisis se ignoran muchas veces los requisitos de espacio del índice asociativo.

12.5.5.2. División recursiva

Si el valor de n_h es mayor o igual que el número de bloque de memoria, la división de las relaciones no se puede hacer en una sola secuencia, puesto que no habría suficientes bloques de memoria intermedia. En lugar de eso, la división se tiene que hacer en varias secuencias. En una se puede dividir la entrada en tantas particiones como bloques de memoria haya disponibles para utilizarlos como memorias intermedias de salida. Cada cajón generado en cada secuencia se lee de manera separada y se divide de nuevo en la siguiente para crear particiones más pequeñas. La función de asociación utilizada en una secuencia es, por supuesto, diferente de la que se ha utilizado en la secuencia anterior. Se repite esta división de la entrada hasta que cada partición de la entrada de construcción quepa en la memoria. Esta división se denomina **división recursiva.**

Una relación no necesita de la división recursiva si $M > n_h + 1$ o, lo que es equivalente, $M > (b_s/M) + 1$, lo cual se simplifica (de manera aproximada) a $M > \sqrt{b_s}$. Por ejemplo, considerando un tamaño de la memoria de 12 megabytes, dividido en bloques de 4 kilobytes, la relación contendría un total de 3K (3.072) bloques. Se puede utilizar una memoria de este tamaño para dividir relaciones de $3K * 3K$ bloques, que son 36 gigabytes. Del mismo modo, una relación del tamaño de 1 gigabyte necesita $\sqrt{256K}$ bloques, o 2 megabytes, para evitar la división recursiva.

12.5.5.3. Gestión de desbordamientos

Se produce el **desbordamiento de una tabla de asociación** en la partición i de la relación de construcción s, si el índice asociativo de s_i es mayor que la memoria principal. El desbordamiento de la tabla de asociación puede producirse si existen muchas tuplas en la relación de construcción con los mismos valores en los atributos de la reunión, o si la función de asociación no tiene las propiedades de aleatoriedad y uniformidad. En cualquier caso, algunas de las particiones tendrán más tuplas que la media, mientras que otras tendrán menos; se dice entonces que la división está **sesgada.**

Se puede controlar parcialmente el sesgo mediante el incremento del número de particiones, de tal manera que el tamaño esperado de cada partición (incluido el índice asociativo en la partición) sea algo menor que el tamaño de la memoria. Por consiguiente, el número de particiones se incrementa en un pequeño valor llamado **factor de escape,** que normalmente es del orden del 20 por ciento del número de particiones calculadas, como se describe en la Sección 12.5.5.

Aunque se sea conservador con los tamaños de las particiones utilizando un factor de escape, todavía pueden producirse desbordamientos. Los desbordamientos de la tabla de asociación se pueden tratar mediante la *resolución del desbordamiento* o la *evitación del desbordamiento*. La **resolución del desbordamiento** se realiza como sigue. Si s_i, para cualquier i, resulta ser demasiado grande, se divide de nuevo en particiones más pequeñas utilizando una función de asociación distinta. Del mismo modo, también se divide r_i utilizando la nueva función de asociación, y solamente es necesario reunir las tuplas en las particiones concordantes.

En cambio, la **evitación del desbordamiento** realiza la división más cuidadosamente, de tal manera que el desbordamiento nunca se produce en la fase de construcción. Para evitar el desbordamiento se implementa la división de la relación de construcción s en muchas particiones pequeñas inicialmente, para luego combinar algunas de estas particiones de tal manera que cada partición combinada quepa en la memoria. Además, la relación de prueba r se tiene que combinar de la misma manera que se combinan las particiones de s, sin importar los tamaños de r_i.

Las técnicas de resolución y evitación podrían fallar en algunas particiones, si un gran número de las tuplas en s tuvieran el mismo valor en los atributos de la reunión. En ese caso, en lugar de crear un índice asociativo en la memoria y utilizar una reunión en bucle anidado para reunir las particiones, se pueden utilizar otras técnicas de reunión, tales como la reunión en bucle anidado por bloques, en esas particiones.

12.5.5.4. Coste de la reunión por asociación

Se considera ahora el coste de una reunión por asociación. El análisis supone que no hay desbordamiento de la tabla de asociación. Primero se considera el caso en el que no es necesaria una división recursiva.

- La división de las dos relaciones r y s necesita una lectura completa de ambas así como su posterior escritura. Esta operación necesita $2(b_r + b_s)$ transferencias de bloques, donde b_r y b_s denotan respectivamente el número de bloques que contienen registros de las relaciones r y s. Las fases de construcción y prueba leen cada una de las particiones una vez, empleando $b_r + b_s$ transferencias adicionales. El número de bloques ocupados por las particiones podría ser ligeramente mayor que $b_r + b_s$ debido a que los bloques están parcialmente ocupados. El acceso a estos bloques llenos en parte puede añadir un gasto adicional de $2n_h$ a lo sumo, ya que cada una de las n_h particiones podría tener un bloque lleno parcialmente que haya que escribir y leer de nuevo. Así, el coste estimado para una reunión por asociación es:

$$3(b_r + b_s) + 4n_h$$

  transferencias de bloques. La sobrecarga $4n_h$ es muy pequeña comparada con $b_r + b_s$ y se puede ignorar.
- Si se da por supuesto que se asignan b_b bloques para la memoria intermedia de entrada y salida, la división necesita $2(\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil)$ búsquedas. Las fases de construcción y prueba requieren solo una búsqueda para cada una de las n_h particiones de cada relación, ya que cada partición se puede leer secuencialmente. Por tanto, la reunión por asociación necesita $2(\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil) + 2n_h$ búsquedas.

Considere ahora el caso en el que es necesario realizar la división recursiva. Cada ciclo reduce el tamaño de cada una de las particiones por un factor esperado de $M-1$; y los ciclos se repiten hasta que cada partición tenga como mucho un tamaño de M bloques. Por tanto, el número esperado de ciclos necesarios para dividir s es $\lceil \log_{M-1}(b_s) - 1 \rceil$.

- Como todos los bloques de s se leen y se escriben en cada ciclo, el número total de transferencias de bloques para dividir s es $2b_s\lceil \log_{M-1}(b_s) - 1 \rceil$. Como el número de ciclos para dividir r es el mismo que el número de ciclos para dividir s, la estimación del coste de la reunión es:

$$2(b_r + b_s)\ \lceil \log_{M-1}(b_s) - 1 \rceil + b_r + b_s$$

  transferencias de bloques.
- De nuevo, dando por supuesto que se asignen b_b bloques en memoria intermedia para cada partición, e ignorando el número relativamente pequeño de búsquedas durante las fases de construcción y prueba, la reunión por asociación con división recursiva necesita:

$$2(\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil)\ \lceil \log_{M-1}(b_s) - 1 \rceil$$

  búsquedas.

Considere, por ejemplo, la reunión *matricula* $\bowtie$ *estudiante*. Con un tamaño de memoria de 20 bloques, se puede dividir *estudiante* en cinco partes, cada una de 20 bloques, con un tamaño tal que quepan en la memoria. Así, solo es necesario un ciclo para la división. De la misma manera, la relación *matricula* se divide en cinco particiones, cada una de 80 bloques. Si se ignora el coste de escribir los bloques parcialmente llenos, el coste es $3(100 + 400) = 1.500$ transferencias de bloques. Hay suficiente memoria para asignar tres bloques a la entrada y cada uno de los cinco bloques de salida durante la división, lo que resulta en $2(\lceil 100/3 \rceil + \lceil 400/3 \rceil) = 336$ búsquedas.

Es posible mejorar la reunión por asociación si el tamaño de la memoria principal es grande. Cuando la entrada de construcción se puede guardar por completo en la memoria principal hay que establecer n_h a 0; de este modo el algoritmo de reunión por asociación se ejecuta rápidamente, sin dividir las relaciones en archivos temporales y sin importar el tamaño de la entrada de prueba. El coste estimado desciende a $b_r + b_s$ transferencias de bloques y dos búsquedas.

12.5.5.5. Reunión por asociación híbrida

El algoritmo de **reunión por asociación híbrida** realiza otra optimización; es útil cuando los tamaños de la memoria son relativamente grandes pero no cabe toda la relación de construcción en la memoria. La fase de división del algoritmo de reunión por asociación necesita un bloque de memoria como memoria intermedia para cada partición que se cree, más un bloque de memoria como memoria intermedia de entrada. Para reducir el impacto de las búsquedas se debería usar un número mayor de bloques como memoria intermedia; sea b_b el número de bloques usados como memoria intermedia para la entrada de cada partición. Por tanto, se necesitan $(n_h + 1) * b_b$ bloques de memoria para dividir las dos relaciones. Si la memoria es mayor que $(n_h + 1) * b_b$, se puede utilizar el resto de la memoria ($M-(n_h + 1) * b_b$ bloques) para guardar en la memoria intermedia la primera partición de la entrada de construcción (es decir, s_0), así que no es necesario escribirla ni leerla de nuevo. Más aún, la función de asociación se diseña de tal manera que el índice asociativo en s_0 quepa en $(M-(n_h + 1) * b_b$ bloques, así que, al final de la división de s, s_0 está en la memoria por completo y se puede construir un índice asociativo en s_0.

Cuando r se divide de nuevo, las tuplas en r_0 no se escriben en disco; en su lugar, según se van generando, el sistema las utiliza para examinar el índice asociativo residente en la memoria de s_0 y para generar las tuplas de salida de la reunión. Después de utilizarlas para la prueba, se descartan las tuplas, así que la partición r_0 no ocupa ningún espacio en la memoria. De este modo se ahorra un acceso de lectura y otro de escritura para cada bloque de r_0 y s_0. Las tuplas en otras particiones se escriben de la manera usual para reunirlas más tarde. El ahorro de la reunión por asociación híbrida puede ser importante si la entrada de construcción es ligeramente mayor que la memoria.

Si el tamaño de la relación de construcción es b_s, n_h es aproximadamente igual a b_s/M. Así, la reunión por asociación híbrida es más útil si $M >> (b_s/M) * b_b$ o si $M >> \sqrt{b_s * b_b}$, donde la notación $>>$ significa *mucho mayor que*. Por ejemplo, suponga que el tamaño del bloque es de 4 kilobytes y que el tamaño de la relación de construcción es de 5 gigabytes y b_b es 20. Entonces, el algoritmo híbrido de reunión por asociación es útil si el tamaño de la memoria es claramente superior a 20 megabytes; las memorias de gigabytes o más son comunes en las computadoras de hoy en día. Si se dedica 1 gigabyte para el algoritmo de reunión, s_0 sería próximo a 1 gigabyte y la reunión por asociación híbrida sería cerca de un 20 por ciento menos costosa que la reunión por asociación.

12.5.6. Reuniones complejas

Las reuniones en bucle anidado y en bucle anidado por bloques son útiles sean cuales sean las condiciones de la reunión. Las otras técnicas de reunión son más eficientes que las reuniones en bucle anidado y sus variantes, aunque solo se pueden utilizar con condiciones simples, tales como las reuniones naturales o las equirreuniones. Se pueden implementar reuniones con condiciones de la reunión más complejas, tales como conjunciones y disyunciones, utilizando las técnicas eficientes de la reunión mediante la aplicación de las técnicas desarrolladas en la Sección 12.3.3 para el manejo de selecciones complejas.

Considere la siguiente reunión con una condición conjuntiva:

$$r \bowtie_{\theta_1 \wedge \theta_2 \wedge \ \ldots \ \wedge \theta_n} s$$

Se pueden aplicar una o más de las técnicas de reunión descritas anteriormente en cada condición individual $r \bowtie_{\theta_1} s$, $r \bowtie_{\theta_2} s$, $r \bowtie_{\theta_3} s$, y así sucesivamente. El resultado total de la reunión se determina calculando primero el resultado de una de estas reuniones simples $r \bowtie_{\theta i} s$; cada par de tuplas del resultado intermedio se compone de una tupla de r y otra de s. El resultado total de la reunión consiste en las tuplas del resultado intermedio que satisfacen el resto de condiciones:

$$\theta_1 \wedge \ldots \wedge \theta_{i-1} \wedge \theta_{i+1} \wedge \ldots \wedge \theta_n$$

Estas condiciones se pueden ir comprobando según se generan las tuplas de $r \bowtie_{\theta i} s$.

Una reunión cuya condición es una disyunción se puede calcular como se indica a continuación. Considere:

$$r \bowtie_{\theta_1 \vee \theta_2 \vee \ldots \vee \theta_n} s$$

La reunión se puede calcular como la unión de los registros de las reuniones $r \bowtie_{\theta_i} s$:

$$(r \bowtie_{\theta_1} s) \cup (r \bowtie_{\theta_2} s) \cup \ldots \cup (r \bowtie_{\theta_n} s)$$

Los algoritmos para calcular la unión de relaciones se describen en la Sección 12.6.

12.6. Otras operaciones

Otras operaciones relacionales y operaciones relacionales extendidas (como eliminación de duplicados, proyección, operaciones sobre conjuntos, reunión externa y agregación) se pueden implementar según se describe en las Secciones 12.6.1 a 12.6.5.

12.6.1. Eliminación de duplicados

Se puede implementar fácilmente la eliminación de duplicados utilizando la ordenación. Las tuplas idénticas aparecerán consecutivas durante la ordenación, pudiéndose eliminar todas las copias menos una. Con la ordenación por mezcla externa se pueden eliminar los duplicados mientras se crea una secuencia antes de que esta se escriba en el disco, reduciendo así el número de bloques transferidos. El resto de duplicados se pueden suprimir durante la etapa de reunión/mezcla, así que el resultado final está libre de repeticiones. El coste estimado en el peor caso para la eliminación de duplicados es el mismo que el coste estimado en el peor caso para la ordenación de una relación.

Se puede implementar también la eliminación de duplicados utilizando la asociación de una manera similar al algoritmo de reunión por asociación. Primero se divide la relación basándose en una función de asociación en la tupla entera. Luego, se lee cada partición, y se construye un índice asociativo en la memoria. Mientras se construye el índice asociativo solo se inserta una tupla únicamente si no estaba ya presente. En otro caso se descarta. Después de que todas las tuplas de la relación se hayan procesado, las tuplas en el índice asociativo se escriben en el resultado. El coste estimado es el mismo que el coste de procesar (división y lectura de cada partición) la relación de construcción en una reunión por asociación.

Debido al coste relativamente alto de la eliminación de duplicados, SQL exige una petición explícita del usuario para suprimir los duplicados; en caso contrario, se conservan.

12.6.2. Proyección

La proyección se puede implementar fácilmente realizando la proyección de cada tupla, pudiendo generar una relación con registros repetidos y suprimiendo después los registros duplicados. La eliminación de duplicados se puede llevar a cabo según se ha descrito en la Sección 12.6.1. Si los atributos de la lista de proyección incluyen una clave de la relación, no se producirán duplicados; por tanto, no será necesario eliminarlos. La proyección generalizada se puede implementar de la misma manera que la proyección.

12.6.3. Operaciones sobre conjuntos

Las operaciones *unión*, *intersección* y *diferencia de conjuntos* se pueden implementar ordenando primero ambas relaciones y examinando después cada relación para producir el resultado. En $r \cup s$, cuando una exploración concurrente en ambas relaciones descubre la misma tupla en los dos archivos, solamente se conserva una de las tuplas. Por otra parte, el resultado de $r \cap s$ contendrá únicamente aquellas tuplas que aparezcan en ambas relaciones. De la misma manera se implementa la *diferencia de conjuntos*, $r-s$, guardando aquellas tuplas de r que no estén en s.

Para todas estas operaciones solamente se necesita una exploración en cada relación de entrada, así que el coste es $b_r + b_s$ transferencias de bloques si las relaciones están ordenadas de igual forma. Suponiendo que en el peor caso solo hubiera un bloque de memoria intermedia para cada relación, se necesitarían un total de $b_r + b_s$ búsquedas, además de las $b_r + b_s$ transferencias de bloques. El número de búsquedas se puede reducir asignando bloques adicionales de memoria intermedia.

Si las relaciones no están ordenadas inicialmente, hay que incluir el coste de la ordenación. Se puede utilizar cualquier otro orden en la evaluación de la operación de conjuntos, siempre que las dos entradas tengan la misma ordenación.

La asociación proporciona otra manera de implementar estas operaciones sobre conjuntos. El primer paso en cada caso es dividir las dos relaciones utilizando la misma función de asociación y de este modo crear las particiones $r_0, r_1, \ldots, r_{nh}$ y $s_0, s_1, \ldots, s_{nh}$. En función de cada operación, el sistema da a continuación estos pasos en cada partición $i = 0, 1, \ldots, n_h$:

- $r \cup s$
  1. Construir un índice asociativo en memoria sobre r_i.
  2. Añadir las tuplas de s_i al índice asociativo solamente si no estaban ya presentes.
  3. Añadir las tuplas del índice asociativo al resultado.
- $r \cap s$
  1. Construir un índice asociativo en memoria en r_i.
  2. Para cada tupla en s_i, probar el índice asociativo y pasar la tupla al resultado únicamente si ya estaba presente en el índice.
- $r - s$
  1. Construir un índice asociativo en memoria en r_i.
  2. Para cada tupla en s_i, probar el índice asociativo y, si la tupla está presente en el índice, suprimirla del índice asociativo.
  3. Añadir las tuplas restantes del índice asociativo al resultado.

12.6.4. Reunión externa

Recordemos las *operaciones de reunión externa* que se describieron en la Sección 4.1.2. Por ejemplo, la reunión natural externa por la izquierda *matricula* ⋈ *estudiante* contiene la reunión de *matricula* y de *estudiante* y, además, para cada tupla t de *matricula* que no concuerde con alguna tupla en *estudiante* (es decir, donde no esté *ID* en *estudiante*), se añade la siguiente tupla t_1 al resultado. Para todos los atributos del esquema de *matricula* la tupla t_1 tiene los mismos valores que la tupla t. El resto de los atributos (del esquema de *estudiante*) de la tupla t_1 contienen el valor nulo.

Se pueden implementar las operaciones de reunión externa empleando una de las dos estrategias siguientes:

1. Calcular la reunión correspondiente, y luego añadir más tuplas al resultado de la reunión hasta obtener la reunión externa resultado. Considere la operación de reunión externa por la izquierda y dos relaciones: r (R) y s (S). Para evaluar $r ⟕_\theta s$, se calcula primero $r \bowtie_\theta s$ y se guarda este resultado como relación temporal q_1. A continuación se calcula $r - \Pi_R(q_1)$, que produce las tuplas de r que no participaron en la reunión. Se puede utilizar cualquier algoritmo para calcular las reuniones. Luego se rellenan cada una de estas tuplas con valores nulos en los atributos de s y se añaden a q_1 para obtener el resultado de la reunión externa.

   La operación reunión externa por la derecha $r ⟖_\theta s$ es equivalente a $s ⟕_\theta r$ y, por tanto, se puede implementar de manera simétrica a la reunión externa por la izquierda. También se puede implementar la operación reunión externa completa $r ⟗_\theta s$ calculando la reunión $r \bowtie s$ y añadiendo luego las tuplas adicionales de las operaciones reunión externa por la izquierda y por la derecha, al igual que antes.

2. Modificar los algoritmos de la reunión. Es fácil extender los algoritmos de reunión en bucle anidado para calcular la reunión externa por la izquierda. Las tuplas de la relación externa que no concuerdan con ninguna tupla de la relación interna se escriben en la salida después de completarlas con valores nulos. Sin embargo, es difícil extender la reunión en bucle anidado para calcular la reunión externa completa.

   Las reuniones externas naturales y las reuniones externas con una condición de equirreunión se pueden calcular mediante extensiones de los algoritmos de reunión por mezcla y reunión por asociación. La reunión por mezcla se puede extender para realizar la reunión externa completa como se muestra a continuación. Cuando se está produciendo la mezcla de las dos relaciones, las tuplas de la relación que no casan con ninguna tupla de la otra relación se pueden completar con valores nulos y escribirse en la salida. Del mismo modo se puede extender la reunión por mezcla para calcular las reuniones externas por la izquierda y por la derecha mediante la copia de las tuplas que no concuerden (rellenadas con valores nulos) desde solamente una de las relaciones. Puesto que las relaciones están ordenadas, es fácil detectar si una tupla casa o no con alguna de las tuplas de la otra relación. Por ejemplo, cuando se hace una reunión por mezcla de *matricula* y *estudiante*, las tuplas se leen según el orden de *ID* y es fácil comprobar para cada tupla si hay alguna tupla coincidente en la otra relación.

   El coste estimado para implementar reuniones externas utilizando el algoritmo de reunión por mezcla es el mismo que para la correspondiente reunión. La única diferencia está en el tamaño del resultado y, por tanto, en los bloques transferidos para copiarlos que no se han tenido en cuenta en las estimaciones anteriores del coste.

   La extensión del algoritmo de reunión por asociación para calcular reuniones externas se deja como ejercicio (Ejercicio 12.15).

12.6.5. Agregación

Recuerde la función de agregación (operador) estudiada en la Sección 3.7. Por ejemplo, la función:

select *nombre_dept*, **avg** (*sueldo*)
from *profesor*
group by *nombre_dept*;

calcula el sueldo medio de cada uno de los departamentos de la universidad.

La operación agregación se puede implementar de una manera parecida a la eliminación de duplicados. Se utiliza la ordenación o la asociación, al igual que se hizo para la supresión de duplicados, pero ahora basándose en la agrupación de atributos (*nombre_dept* en el ejemplo anterior). Sin embargo, en lugar de eliminar las tuplas con el mismo valor en los atributos de la agrupación, se reúnen en grupos y se aplican las operaciones agregación en cada grupo para obtener el resultado.

El coste estimado para la implementación de la operación agregación es el mismo que en la eliminación de duplicados para las funciones de agregación como **min, max, sum, count** y **avg.**

En lugar de reunir todas las tuplas en grupos y aplicar entonces las funciones de agregación, se pueden implementar las funciones de agregación **sum, min, max, count** y **avg** sobre la marcha según se construyen los grupos. Para el caso de **sum, min** y **max,** cuando se encuentran dos tuplas del mismo grupo el sistema las sustituye por una sola tupla que contenga el cálculo de **sum, min** o **max,** respectivamente, de las columnas que se están agregando. Para la operación **count,** se mantiene una cuenta incremental para cada grupo con las tuplas añadidas. Por último, la operación **avg** se implementa calculando la suma y la cuenta de valores sobre la marcha, para dividir finalmente la suma entre la cuenta para obtener la media.

Si todas las tuplas del resultado caben en la memoria, las implementaciones basadas en ordenación y las basadas en asociación no necesitan escribir ninguna tupla en el disco. Según se leen las tuplas se pueden insertar en una estructura ordenada en árbol o en un índice asociativo. Así, cuando se utilizan las técnicas de agregación sobre la marcha, solamente es necesario almacenar una tupla para cada uno de los grupos. Por tanto, la estructura ordenada en árbol o el índice asociativo caben en la memoria y se puede procesar la agregación con solo b_r transferencias de bloques (y una búsqueda), en lugar de las $3b_r$ transferencias (y en el peor caso de hasta $2b_r$ búsquedas) que se necesitarían en otro caso.

12.7. Evaluación de expresiones

Hasta este momento se ha estudiado cómo llevar a cabo operaciones relacionales individuales. Ahora se considera cómo evaluar una expresión que contiene varias operaciones. La manera más evidente de evaluar una expresión es simplemente evaluar una operación a la vez en un orden apropiado. El resultado de cada evaluación se **materializa** en una relación temporal para su inmediata utilización. Un inconveniente de esta aproximación es la necesidad de construir relaciones temporales, que (a menos que sean pequeñas) se tienen que escribir en disco. Un enfoque alternativo es evaluar varias operaciones de manera simultánea en un **cauce** (*pipeline*), con los resultados de una operación que se pasan a la siguiente, sin la necesidad de almacenar relaciones temporales.

En las Secciones 12.7.1 y 12.7.2 se consideran ambos enfoques, *materialización* y *encauzamiento*. Se verá que el coste de estos enfoques puede diferir sustancialmente, pero también que existen casos en los que solo es posible la materialización.

12.7.1. Materialización

Intuitivamente es más fácil entender cómo evaluar una expresión observando una representación gráfica de la expresión en un **árbol de operadores.** Considere la expresión:

$$\Pi_{nombre}\,(\sigma_{edificio\,=\,\text{«Watson»}}\,(departamento) \bowtie profesor)$$

de la Figura 12.11.

Si se aplica el enfoque de la materialización, se comienza por las operaciones de la expresión de nivel más bajo (de la parte inferior

del árbol). En el ejemplo solamente hay una de estas operaciones: la operación selección en *departamento*. Las entradas de las operaciones de nivel más bajo son las relaciones de la base de datos. Se ejecutan estas operaciones utilizando los algoritmos ya estudiados y almacenando sus resultados en relaciones temporales. Luego se utilizan estas relaciones temporales para ejecutar las operaciones del siguiente nivel en el árbol, cuyas entradas son ahora o bien relaciones temporales, o bien relaciones almacenadas en la base de datos. En este ejemplo, las entradas de la reunión son la relación *profesor* y la relación temporal producida por la selección en *departamento*. Ahora se puede evaluar la reunión creando otra relación temporal.

Repitiendo este proceso se calcularía finalmente la operación en la raíz del árbol, obteniendo el resultado final de la expresión. En el ejemplo se consigue el resultado final mediante la ejecución de la operación proyección de la raíz utilizando como entrada la relación temporal creada por la reunión.

Una evaluación como la descrita se denomina **evaluación materializada,** puesto que los resultados de cada operación intermedia se crean (materializan) para utilizarse a continuación en la evaluación de las operaciones del siguiente nivel.

El coste de una evaluación materializada no es simplemente la suma de los costes de las operaciones involucradas. Cuando se calcularon los costes estimados de los algoritmos se ignoró el coste de escribir el resultado de la operación en el disco. Para calcular el coste de evaluar una expresión como la que se ha hecho hay que añadir los costes de todas las operaciones, incluyendo el coste de escribir los resultados intermedios en el disco. Supondremos que los registros del resultado se acumulan en una memoria intermedia y que cuando esta se llena, los registros se escriben en el disco. El número de bloques escritos, b_r, se puede estimar en n_r/f_r, donde n_r es el número aproximado de tuplas de la relación resultado r y f_r es el *factor de bloques* de la relación resultado, es decir, el número de registros de r que caben en un bloque. Además del tiempo de transferencia es posible que se necesiten algunas búsquedas, ya que la cabeza del disco se puede haber desplazado entre escrituras sucesivas. Se puede estimar el número de búsquedas como $\lceil b_r/b_b \rceil$, donde b_b es el tamaño en bloques de la memoria intermedia para la salida (medida en bloques).

La **memoria intermedia doble** (usando dos memorias intermedias, una donde progresa la ejecución del algoritmo mientras que la otra se está copiando al disco) permite que el algoritmo se ejecute más rápidamente mediante la ejecución en paralelo de acciones en la CPU con acciones de E/S. El número de búsquedas se puede reducir asignando bloques adicionales en la memoria intermedia de salida y escribiendo varios bloques a la vez.

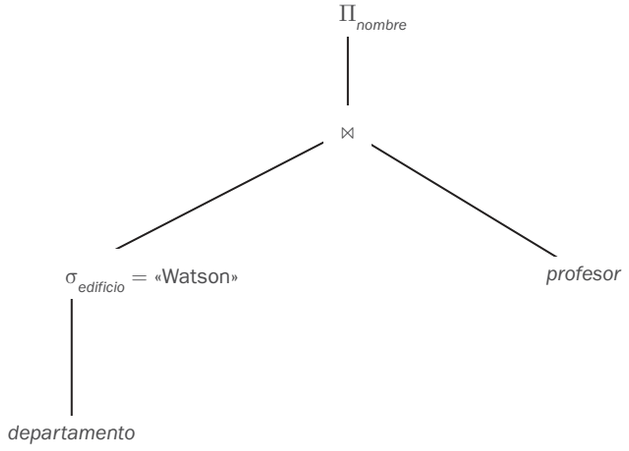


Figura 12.11. Representación gráfica de una expresión.

12.7.2. Encauzamiento

Se puede mejorar la eficiencia de la evaluación de consultas mediante la reducción del número de archivos temporales que se producen. Esta reducción se lleva a cabo mediante la combinación de varias operaciones relacionales en un *encauzamiento* de operaciones, en el que se pasan los resultados de una operación a la siguiente operación del encauzamiento. Esta evaluación, como se ha descrito, se denomina **evaluación encauzada.**

Por ejemplo, considere la expresión $(\Pi_{a1,a2}(r \bowtie s))$. Si se aplicara la materialización, la evaluación implicaría la creación de una relación temporal para guardar el resultado de la reunión y la posterior lectura del resultado para realizar la proyección. Estas operaciones se pueden combinar como sigue. Cuando la operación reunión genera una tupla del resultado, se pasa inmediatamente esa tupla al operador de proyección para su procesamiento. Mediante la combinación de la reunión y de la proyección, se evita la creación de resultados intermedios, creando en su lugar el resultado final directamente.

La creación de un encauzamiento de operaciones tiene dos ventajas:

1. Elimina el coste de leer y escribir las relaciones temporales, reduciendo el coste de la evaluación de consultas.
2. Puede empezar a generar resultados de la consulta rápidamente, si el operador raíz del plan de evaluación se combina en un encauzamiento con las entradas. Esto puede resultar muy útil si los resultados se muestran al usuario según se generan ya que, en caso contrario, puede haber un gran retardo antes de que el usuario vea algún resultado.

12.7.2.1. Implementación del encauzamiento

Se puede implementar el encauzamiento construyendo una única y compleja operación que combine las operaciones que constituyen el encauzamiento. Aunque este enfoque podría ser factible en muchas situaciones, en general es deseable reutilizar el código en operaciones individuales en la construcción del encauzamiento.

En el ejemplo de la Figura 12.11, las tres operaciones se pueden situar en un encauzamiento, en el que los resultados de la selección se pasan a la reunión según se generan. Por su parte, los resultados de la reunión se envían a la proyección según se van generando. Así, los requisitos de memoria son bajos, ya que los resultados de una operación no se almacenan por mucho tiempo. Sin embargo, como resultado del encauzamiento, las entradas de las operaciones no están disponibles todas a la vez para su procesamiento.

Los encauzamientos se pueden ejecutar de alguno de los siguientes dos modos:

1. En un **encauzamiento bajo demanda,** el sistema reitera peticiones de tuplas desde la operación de la cima del encauzamiento. Cada vez que una operación recibe una petición de tuplas, calcula la siguiente tupla (o tuplas) a devolver y la envía. Si las entradas de la operación no están encauzadas, se calcula(n) la(s) siguiente(s) tupla(s) a devolver de las relaciones de entrada mientras se lleva la cuenta de lo que se ha remitido hasta el momento. Si alguna de sus entradas está encauzada, la operación también hace peticiones de tuplas desde sus entradas encauzadas. Así, utilizando las tuplas recibidas en sus entradas encauzadas, la operación calcula sus tuplas de salida y las envía hasta su progenitora.
2. En un **encauzamiento dirigido por los productores,** las operaciones no esperan a que se produzcan peticiones para producir tuplas, en su lugar generan las tuplas **impacientemente.** Cada operación en el encauzamiento dirigido por los productores se modela como un proceso separado o un hilo del sistema que coge un flujo de tuplas de sus entradas y genera un flujo de tuplas en su salida en un encauzamiento.

Más adelante se describe cómo se pueden implementar los encauzamientos bajo demanda y dirigidos por los productores.

Cada operación en un encauzamiento bajo demanda se puede implementar como un **iterador,** que proporciona las siguientes funciones: *abrir*( ) (*open*), *siguiente*( ) (*next*) y *cerrar*( ) (*close*). Después de una llamada a *abrir*( ), cada llamada a *siguiente*( ) devuelve la siguiente tupla de salida de la operación. La implementación de la operación realiza por turno llamadas a *abrir*( ) y a *siguiente*( ) desde sus entradas cuando necesita tuplas de entrada. La función *cerrar*( ) comunica al iterador que no necesita más tuplas. De este modo, el iterador mantiene el **estado** de su ejecución entre las llamadas, de tal manera que las sucesivas peticiones *siguiente*( ) reciban sucesivas tuplas resultado.

Por ejemplo, en un iterador que implemente la operación selección usando la búsqueda lineal, la operación *abrir*( ) inicia una exploración de archivo y el estado del iterador registra el punto hasta el que el archivo se ha explorado. Cuando se llama a la función *siguiente*( ), la exploración del archivo continúa después del punto anterior; cuando se encuentre la siguiente tupla que cumpla la selección al explorar el archivo, se devuelve la tupla después de almacenar el punto en el que se encontró en el estado del iterador. Una operación *abrir*( ) del iterador de reunión por mezcla abriría sus entradas y, si aún no están ordenadas, también las ordenaría. En las llamadas a *siguiente*( ) se devolvería el siguiente par de tuplas coincidentes. La información de estado consistiría en el grado en que se ha explorado cada entrada. Los detalles de la implementación de los iteradores se dejan para completar en el Ejercicio práctico 12.7.

Los encauzamientos dirigidos por los productores, por otra parte, se implementan de forma diferente. Para cada par de operaciones adyacentes en un encauzamiento dirigido por los productores, el sistema crea una memoria intermedia para mantener las tuplas que pasan de una operación a la siguiente. Los procesos o hilos que ejecutan las distintas operaciones se ejecutan concurrentemente. Cada operación desde el fondo del encauzamiento genera continuamente tuplas y las pone en la memoria intermedia de salida, hasta que se llena. Una operación de cualquier otro nivel del encauzamiento genera tuplas de salida cuando recoge tuplas de su entrada de un nivel inferior, hasta que se llena la memoria intermedia de salida. Cuando una operación de otro nivel usa una tupla de su entrada, la elimina de la memoria intermedia de su entrada. En cualquier caso, cuando la memoria intermedia de salida se llena, la operación espera hasta que la operación progenitora elimina tuplas, de forma que la memoria intermedia tenga sitio en el que volver a insertar más. En este momento la operación genera más tuplas, hasta que la memoria intermedia se llena de nuevo. La operación repite el proceso hasta que se han generado todas las tuplas de salida.

El sistema necesita cambiar de una operación a otra solamente cuando se llena una memoria intermedia de salida o cuando una memoria intermedia de entrada está vacía y se necesitan más tuplas para generar las tuplas de salida. En un sistema de procesamiento paralelo, las operaciones de encauzamiento se pueden ejecutar concurrentemente en distintos procesadores (consulte el Capítulo 18).

Se puede pensar en el encauzamiento dirigido por los productores como una **inserción** de datos de abajo arriba en el árbol de operaciones, mientras que el encauzamiento bajo demanda se puede pensar como una **extracción** de datos desde la cima del árbol de operaciones. Mientras que las tuplas se generan *impacientemente* en el encauzamiento por los productores, se generan de forma **perezosa,** bajo demanda, en el encauzamiento bajo demanda. Es más común el uso del encauzamiento bajo demanda, ya que es más sencillo de implementar. Sin embargo, el encauzamiento dirigido por los productores es muy útil en sistemas de procesamiento paralelo.

12.7.2.2. Algoritmos de evaluación para el encauzamiento

Algunas operaciones, como la de ordenación, son inherentemente **operaciones bloqueantes,** es decir, no se puede empezar a generar tuplas de salida hasta que se han explorado todas las tuplas de la entrada.[5]

Otras operaciones como la reunión no son inherentemente bloqueantes, pero el algoritmo de evaluación concreto puede ser bloqueante. Por ejemplo, el algoritmo de reunión por asociación es una operación bloqueante, ya que se necesita que se recuperen y realicen particiones ambas entradas, antes de poder dar salida a ninguna tupla. Por otra parte, el algoritmo de reunión en bucle anidado indexada puede generar tuplas según las obtiene de la relación externa. Se dice en este sentido que se encuentra **encauzada** en su relación externa (por la izquierda), aunque sea bloqueante en su entrada indexada (por la derecha), ya que hay que construir el índice completo antes de que el algoritmo de reunión en bucle anidado indexada se pueda ejecutar.

La reunión por asociación híbrida se puede ver como parcialmente encauzada sobre la relación de prueba, ya que puede generar tuplas de salida de la primera partición según se reciben de la relación de prueba. Sin embargo, las tuplas que no estén en la primera partición solo se generarán después de que se haya recibido la relación de entrada encauzada completa. La reunión por asociación híbrida proporciona una evaluación encauzada sobre su entrada de prueba, si la entrada construida cabe completamente en la memoria, o medio encauzada, si la mayor parte de la entrada construida cabe en la memoria.

Si ambas entradas se ordenan por el atributo de la reunión y la condición de reunión es una equirreunión, se puede usar una reunión por mezcla, con sus dos entradas encauzadas.

```
hecho_r := false;
hecho_s := false;
r := ∅;
s := ∅;
resultado := ∅;
while not hecho_r or not hecho_s do
  begin
    if la cola está vacía then esperar hasta que la cola no esté vacía;
    t := entrada de la cima de la cola;
    if t = Fin_r then hecho_r := true
      else if t = Fin_s then hecho_s := true
        else if t es de la entrada r
          then
            begin
              r := r ∪ {t};
              resultado:= resultado ∪ ({t} ⋈ s);
            end
          else /* t es de la entrada s */
            begin
              s := s ∪ {t};
              resultado:= resultado ∪ (r ⋈ {t});
            end
  end
```

Figura 12.12. Algoritmo de reunión encauzada doble.

5 Las operaciones bloqueantes, como la ordenación, pueden generar tuplas pronto si se sabe que la entrada satisface algunas propiedades especiales, como que ya esté ordenada, o parcialmente ordenada. Sin embargo, si no se conoce esta información, las operaciones bloqueantes no pueden generar tuplas pronto.

Sin embargo, en el caso más común en que se quiere que las dos entradas que queremos encauzar en la reunión no estén ordenadas, la otra alternativa es la técnica de **reunión encauzada doble**, que se muestra en la Figura 12.12. El algoritmo supone que las tuplas de entrada de ambas relaciones, r y s, se encauzan. Las tuplas de ambas relaciones se encolan para su procesamiento en una única cola. Se insertan entradas especiales en la cola, llamadas End_r y End_s, que se usan para marcar el fin del archivo, después de que se hayan generado todas las tuplas de r y de s. Según se añaden tuplas a r y a s, hay que actualizar los índices. Cuando los índices se usan con r y s, el algoritmo resultado se denomina técnica de **reunión por asociación encauzada doble.**

El algoritmo de reunión por asociación encauzada doble de la Figura 12.12 supone que ambas entradas caben en la memoria. En el caso de que las entradas fuesen mayores que la memoria, aún es posible usar esta técnica de la forma usual hasta que se llene la memoria disponible. Cuando la memoria disponible se llena, las tuplas r y s que hayan llegado hasta ese punto se pueden tratar como si estuviesen en las particiones r_0 y s_0, respectivamente. A las tuplas de r y s que lleguen después se les asigna las particiones r_1 y s_1, respectivamente, que se escriben a disco, y no se añaden al índice en memoria. Sin embargo, las tuplas asignadas a r_1 y s_1 se usan para probar con s_0 y r_0, respectivamente, antes de que se escriban en disco. Por tanto, la reunión de r_1 con s_0, y de s_0 con r_1, también se lleva a cabo de forma encauzada. Tras haber procesado r y s, se puede llevar a cabo la reunión de las tuplas r_1 con las tuplas s_1, para completar la reunión; cualquiera de las técnicas de reunión que se han tratado anteriormente se pueden utilizar para realizar la reunión de r_1 con s_1.

12.8. Resumen

- La primera acción que el sistema debe realizar en una consulta es traducirla en su formato interno, que (para sistemas de bases de datos relacionales) normalmente está basado en el álgebra relacional. En el proceso de generación del formato interno de la consulta, el analizador comprueba la sintaxis, verifica que los nombres de la relación que figuran en la consulta son nombres de relaciones de la base de datos, etc. Si la consulta se expresó en términos de una vista, el analizador sustituye todas las referencias al nombre de la vista con su expresión del álgebra relacional para calcularla.
- Dada una consulta, generalmente hay diversos métodos para calcular la respuesta. Es responsabilidad del sistema transformar la consulta proporcionada por el usuario en otra consulta equivalente que se pueda ejecutar de un modo más eficiente. En el Capítulo 13 se estudia la optimización de consultas.
- Las consultas que impliquen selecciones sencillas se pueden procesar mediante una búsqueda lineal, una búsqueda binaria o utilizando índices. Así mismo, se pueden manejar selecciones más complejas mediante uniones e intersecciones de los resultados de selecciones simples.
- Las relaciones que sean más grandes que la memoria se pueden ordenar utilizando el algoritmo de ordenación por mezcla externa.
- Las consultas que impliquen una reunión natural se pueden procesar de varias maneras, dependiendo de la disponibilidad de índices y del tipo de almacenamiento físico utilizado para las relaciones.
  - Si la reunión resultante es casi tan grande como el producto cartesiano de las dos relaciones, una estrategia de *reunión en bucle anidado por bloques* podría ser ventajosa.
  - Si hay índices disponibles, se puede utilizar *la reunión en bucle anidado indexada*.
  - Si las relaciones están ordenadas, sería deseable una *reunión por mezcla*. Además, podría ser útil ordenar una relación antes que calcular una reunión (para permitir el uso de una estrategia de reunión por mezcla).
  - El algoritmo de *reunión por asociación* divide la relación en varias particiones, de tal manera que cada partición de una de las relaciones quepa en la memoria. La división se lleva a cabo con una función de asociación en los atributos de la reunión, de tal modo que los pares de particiones correspondientes se puedan reunir independientemente.
- La eliminación de duplicados, la proyección y las operaciones de conjuntos (unión, intersección y diferencia) se pueden realizar mediante ordenación o asociación.
- Las operaciones de reunión externa se pueden implementar como extensiones simples de los algoritmos de reunión.
- Las técnicas de asociación y ordenación son duales, en el sentido de que muchas operaciones como la eliminación de duplicados, la agregación, las reuniones y las reuniones externas que se pueden implementar mediante asociación también se pueden implementar por ordenación, y viceversa; es decir, cualquier operación que se pueda implementar con ordenación también puede serlo por asociación.
- Una expresión se puede evaluar mediante materialización, en la que el sistema calcula el resultado de cada subexpresión y lo almacena en disco, y después lo usa para calcular el resultado de la expresión progenitora.
- El encauzamiento ayuda a evitar la escritura en disco de los resultados de muchas subexpresiones usando los resultados de la expresión progenitora según se van generando.

Términos de repaso

- Procesamiento de consultas.
- Primitiva de evaluación.
- Plan de ejecución de consultas.
- Plan de evaluación de consultas.
- Motor de ejecución de consultas.
- Medidas del coste de las consultas.
- E/S secuencial.
- E/S aleatoria.
- Exploración de archivos.
- Búsqueda lineal.
- Selecciones usando índices.
- Rutas de acceso.

- Exploración de índices.
- Selección conjuntiva.
- Selección disyuntiva.
- Índice compuesto.
- Intersección de identificadores.
- Ordenación externa.
- Ordenación por mezcla externa.
- Secuencias.
- Mezcla de *N* vías.
- Equirreunión.
- Reunión en bucle anidado.
- Reunión en bucle anidado por bloques.
- Reunión en bucle anidado indexada.
- Reunión por mezcla.
- Reunión por ordenación por mezcla.
- Reunión por mezcla híbrida.
- Reunión por asociación.
  - Construir.
  - Prueba.
  - Entrada de construcción.
  - Entrada de prueba.
  - División recursiva.
  - Desbordamiento de la tabla de asociación.
  - Sesgo.
  - Factor de escape.
  - Resolución del desbordamiento.
  - Evitación del desbordamiento.
- Reunión por asociación híbrida.
- Árbol de operadores.
- Evaluación materializada.
- Memoria intermedia doble.
- Evaluación encauzada.
  - Cauce bajo demanda (perezoso, extracción).
  - Cauce por productor (impaciente, inserción).
  - Iterador.
- Reunión encauzada doble.

Ejercicios prácticos

12.1. Suponga (para simplificar este ejercicio) que solo cabe una tupla en un bloque y que la memoria puede contener como máximo tres bloques. Indique las secuencias creadas en cada secuencia del algoritmo de ordenación por mezcla cuando se aplica para ordenar por el primer atributo de las siguientes tuplas: (canguro, 17), (ualabí, 21), (emú, 1), (wombat, 13), (ornitorrinco, 3), (león, 8), (jabalí, 4), (cebra, 11), (koala, 6), (hiena, 9), (cálao, 2), (babuino, 12).

12.2. Considere la base de datos del banco de la Figura 12.13, en la que se ha subrayado la clave primaria, y las siguientes consultas de SQL:

select *T.nombre_sucursal*
from *sucursal T*, *sucursal S*
where *T.activos* > *S.activos* **and** *S.ciudad_sucursal* = «Brooklyn»

Escriba una expresión del álgebra relacional equivalente a la ofrecida que sea más eficiente. Justifique la elección.

sucursal (*nombre_sucursal*, *ciudad_sucursal*, *activos*)
cliente (*nombre_cliente*, *calle_cliente*, *ciudad_cliente*)
préstamo (*número_préstamo*, *nombre_sucursal*, *cantidad*)
prestador (*nombre_cliente*, *número_préstamo*)
cuenta (*número_cuenta*, *nombre_sucursal*, *saldo*)
depositante (*nombre_cliente*, *número_cuenta*)

Figura 12.13. Base de datos de un banco.

12.3. Dadas las relaciones $r_1(A, B, C)$ y $r_2(C, D, E)$ con las siguientes propiedades: r_1 tiene 20.000 tuplas, r_2 tiene 45.000 tuplas, en cada bloque caben, como máximo, 25 tuplas de r_1 o 30 tuplas de r_2. Estime el número de transferencias de bloques y de búsquedas necesarias utilizando las siguientes estrategias para la reunión $r_1 \bowtie r_2$:

a. Reunión en bucle anidado.
b. Reunión en bucle anidado por bloques.
c. Reunión por mezcla.
d. Reunión por asociación.

12.4. El algoritmo de reunión en bucle anidado indexada descrito en la Sección 12.5.3 puede ser ineficiente si el índice es secundario y existen varias tuplas con el mismo valor en los atributos de la reunión. ¿Por qué es ineficiente? Describa una forma de reducir el coste de recuperar las tuplas de la relación más interna utilizando la ordenación. ¿Bajo qué condiciones sería este algoritmo más eficiente que la reunión por mezcla híbrida?

12.5. Sean r y s dos relaciones sin índices que no están ordenadas. Suponiendo una memoria infinita, ¿cuál es la forma de menor coste (en términos de operaciones de E/S) para calcular $r \bowtie s$? ¿Cuánta memoria se necesita en este algoritmo?

12.6. Considere la base de datos del banco de la Figura 12.13, donde la clave primaria se ha subrayado. Suponga que hay un índice de árbol B^+ disponible en *ciudad_sucursal* de la relación *sucursal* y que no hay más índices. ¿Cuál sería el mejor modo de manejar las siguientes selecciones con negaciones?

a. $\sigma_{\neg(ciudad_sucursal < \text{«Brooklyn»})}(sucursal)$
b. $\sigma_{\neg(ciudad_sucursal = \text{«Brooklyn»})}(sucursal)$
c. $\sigma_{\neg(ciudad_sucursal = \text{«Brooklyn»}) \vee assets < 5000)}(sucursal)$

12.7. Escriba el pseudocódigo para un iterador que implemente la reunión en bucle anidado indexada, donde la relación externa esté encauzada. Defina las funciones iteradoras estándar *open*(), *next*() y *close*(). Indique la información de estado del iterador que se debe guardar entre las llamadas.

12.8. Diseñe algoritmos basados en ordenación y asociación para el cálculo de la operación división relacional (consulte los ejercicios prácticos del Capítulo 6 para ver una definición de la operación división).

12.9. ¿Cuál es el efecto sobre el coste de la mezcla de secuencias si el número de bloques de memoria intermedia se incrementa en cada secuencia mientras se mantiene la memoria total disponible para la memoria intermedia de las secuencias?

Ejercicios

12.10. Suponga que se necesita ordenar una relación de 40 gigabytes, con bloques de 4 kilobytes, usando una memoria de 40 megabytes. Suponga que el coste de una búsqueda es de 5 milisegundos, mientras que la velocidad de transferencia de disco es de 40 megabytes por segundo.

a. Calcule el coste de ordenar la relación, en segundos, con $b_b = 1$ y $b_b = 100$.

b. Para cada uno de los casos, ¿cuántos pasos de mezcla se necesitan?

c. Suponga que se utiliza un dispositivo de memoria flash en lugar de un disco, y que el tiempo de búsqueda es de 1 milisegundo, y la velocidad de transferencia de 40 megabytes por segundo. Calcule de nuevo el coste de la ordenación de la relación, en segundos, con $b_b = 1$ y con $b_b = 100$.

12.11. Considere los siguientes operadores extendidos del álgebra relacional. Describa cómo implementar cada uno de estos operadores usando la ordenación y la asociación.

a. Semiunión ($\ltimes_\theta$): $r \ltimes_\theta s$ se define como $\Pi_R(r \bowtie_\theta s)$, donde R es el conjunto de atributos en el esquema de r; es decir, que selecciona aquellas tuplas r_i en r para las que existe una tupla s_j en s tal que r_i y s_j satisfacen el predicado θ.

b. Anti-semiunión ($\bar{\ltimes}_\theta$): $r\ 1\ \bar{>} \ltimes_\theta s$ se define como $r - \Pi_R(r \bowtie_\theta s)$; es decir, que selecciona aquellas tuplas r_i en r para las que no existe una tupla s_j en s tal que r_i y s_j satisfacen el predicado θ.

12.12. ¿Por qué no hay que obligar a los usuarios a que elijan explícitamente una estrategia de procesamiento de la consulta? ¿Hay casos en los que *es* deseable que los usuarios sepan el coste de las distintas estrategias posibles? Razone la respuesta.

12.13. Diseñe una variante del algoritmo híbrido de reunión por mezcla para el caso de que las dos relaciones no estén ordenadas según el orden físico de almacenamiento, pero ambas tengan un índice secundario ordenado en los atributos de la reunión.

12.14. Estime el número de accesos a bloques que necesita la solución del Ejercicio 12.13 para $r_1 \bowtie r_2$ donde r_1 y r_2 son como las relaciones definidas en el Ejercicio práctico 12.3.

12.15. El algoritmo de reunión por asociación descrito en la Sección 12.5.5 calcula la reunión natural de dos relaciones. Describa cómo extender el algoritmo de reunión por asociación para calcular la reunión externa por la izquierda, la reunión externa por la derecha y la reunión externa completa. (Sugerencia: se puede mantener información adicional con cada tupla en el índice asociativo para detectar si alguna tupla en la relación de prueba concuerda con alguna tupla del índice asociativo). Compruebe el algoritmo con las relaciones *matricula* y *estudiante*.

12.16. Se usa encauzamiento para evitar escribir resultados intermedios a disco. Suponga que se necesita ordenar una relación r usando la ordenación por mezcla y reuniendo por mezcla el resultado con una relación s previamente ordenada.

a. Describa cómo se puede encauzar la salida de la ordenación de r con la reunión por mezcla sin escribir a disco.

b. La misma idea es aplicable incluso si ambas entradas a la reunión por mezcla son las salidas de las operaciones de ordenación por mezcla. Sin embargo, la memoria disponible tiene que compartirse entre las dos operaciones de mezcla (el propio algoritmo de reunión por mezcla necesita muy poca memoria). ¿Cuál es el efecto de tener que compartir la memoria sobre el coste de cada operación de ordenación por mezcla?

12.17. Escriba el pseudocódigo para un iterador que implemente una versión del algoritmo de mezcla por ordenación en el que el resultado de la mezcla final se encauza a su consumidor. El pseudocódigo debe definir las funciones iteradoras estándar *abrir*(), *siguiente*() y *cerrar*(). Muestre la información de estado del iterador que se debe guardar entre las llamadas.

12.18. Suponga que hay que calcular ${}_A\mathcal{G}_{sum(C)}(r)$ y ${}_{A,B}\mathcal{G}_{sum(C)}(r)$. Describa cómo calcular ambas juntas usando una única ordenación de r.

Notas bibliográficas

Todos los procesadores de consultas deben analizar instrucciones del lenguaje de consulta y deben traducirlas a su formato interno. El análisis de los lenguajes de consultas difiere poco del análisis de los lenguajes de programación tradicionales. La mayoría de los libros sobre compiladores tratan las principales técnicas de análisis y presentan la optimización desde el punto de vista de los lenguajes de programación.

Graefe y McKenna [1993b] presentan una excelente revisión de las técnicas de evaluación de consultas.

Knuth [1973] presenta una excelente descripción de algoritmos de ordenación externa, incluyendo una optimización denominada *selección de reemplazamiento* que puede originar secuencias iniciales que son (de media) el doble del tamaño de la memoria. Nyberg et ál. [1995] han demostrado que debido al mal comportamiento de la caché del procesador, la selección de reemplazamiento se comporta peor que el ordenamiento rápido en memoria para la generación de secuencias, eliminando las ventajas de la generación de secuencias más grandes. Nyberg et ál. [1995] presentan un algoritmo eficiente de ordenación externa que considera los efectos de la caché del procesador. Los algoritmos de evaluación de consultas que consideran los efectos de la caché se han estudiado ampliamente; véase, por ejemplo, Harizopoulos y Ailamaki [2004].

De acuerdo con estudios del rendimiento realizados a mediados de los años setenta del siglo XX, los sistemas de bases de datos de esa época solamente utilizaban reunión en bucle anidado y reunión por mezcla. Estos estudios, que estuvieron relacionados con el desarrollo de System R, determinaron que tanto la reunión en bucle anidado como la reunión por mezcla casi siempre proporcionaban el método de reunión óptimo (Blasgen y Eswaran [1976]). Por tanto, estos son los dos únicos algoritmos de reunión implementados en System R. Sin embargo, Blasgen y Eswaran [1976] no incluyeron el análisis de los algoritmos de reunión por asociación. Actualmente, estos algoritmos se consideran muy eficientes y se usan mucho.

Los algoritmos de reunión por asociación se desarrollaron inicialmente para sistemas de bases de datos paralelos. La técnica de reunión por asociación híbrida se describe en Shapiro [1986]. Zeller y Gray [1990], y Davison y Graefe [1994] describen técnicas de reunión por asociación que se pueden adaptar a la memoria disponible, lo que es importante en sistemas en los que se pueden ejecutar a la vez varias consultas. Graefe et ál. [1998] describen el uso de las reuniones por asociación y los *equipos asociados*, que permiten el encauzamiento de las reuniones por asociación usando la misma división para todas las reuniones por asociación en una secuencia encauzada, en Microsoft SQL Server.

Optimización de consultas 13

La **optimización de consultas** es el proceso de selección del plan de evaluación de las consultas más eficiente de entre las muchas estrategias generalmente disponibles para el procesamiento de una consulta dada, especialmente si la consulta es compleja. No se espera que los usuarios escriban las consultas de modo que puedan procesarse de manera eficiente. Por el contrario, se espera que el sistema cree un plan de evaluación que minimice el coste de la evaluación de las consultas. Ahí es donde entra en acción la optimización de consultas.

Un aspecto de la optimización de las consultas tiene lugar en el nivel del álgebra relacional, donde el sistema intenta hallar una expresión que sea equivalente a la expresión dada, pero de ejecución más eficiente. Otro aspecto es la elección de una estrategia detallada para el procesamiento de la consulta, como puede ser la selección del algoritmo que se usará para ejecutar una operación, la selección de los índices concretos que se van a emplear, etc.

La diferencia en coste (en términos de tiempo de evaluación) entre una estrategia buena y una mala suele ser sustancial, y puede resultar de varios órdenes de magnitud. Por tanto, merece la pena que el sistema invierta una cantidad importante de tiempo en la selección de una buena estrategia para el procesamiento de la consulta, aunque esa consulta solo se ejecute una vez.

13.1. Visión general

Considere la expresión del álgebra relacional para la consulta «Encontrar los nombres de todos los profesores del departamento de Música junto con los nombres de todas las asignaturas que enseñan».

$$\Pi_{nombre,nombre_asig}(\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor \bowtie (enseña \bowtie \Pi_{id_asignatura,nombre_asig}(asignatura))))$$

Adviértase que la proyección de *asignatura* en (*id_asignatura*, *nombre_asig*) es necesaria, ya que *asignatura* comparte el atributo *nombre_dept* con *profesor*; si no se eliminase este atributo usando la proyección, la expresión anterior que usa la reunión natural solo devolvería los cursos del departamento de Música, incluso aunque algunos profesores del departamento de Música enseñasen asignaturas en otros departamentos.

La expresión anterior crea una relación intermedia de gran tamaño:

$$profesor \bowtie enseña \bowtie \Pi_{id_asignatura,nombre_asig}(asignatura)$$

Sin embargo, solo resultan de interés unas pocas tuplas de esta relación (las correspondientes a los profesores del departamento de Música), y solo en dos de los diez atributos de la relación. Dado que solo son relevantes las tuplas de la relación *profesor* que pertenecen al departamento de Música, no hace falta considerar las tuplas que no cumplen *nombre_dept* = «Música». Al reducir el número de tuplas de la relación *profesor* a las que es necesario acceder, se reduce el tamaño del resultado intermedio. La consulta queda ahora representada por la siguiente expresión del álgebra relacional:

$$\Pi_{nombre,nombre_asig}((\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor)) \bowtie (enseña \bowtie \Pi_{id_asignatura,nombre_asig}(asignatura)))$$

que es equivalente a la expresión algebraica original, pero genera relaciones intermedias de menor tamaño. La Figura 13.1 muestra la expresión inicial y la transformada.

Un plan de evaluación define exactamente qué algoritmo utilizar para cada operación, y cómo se debe coordinar la ejecución de las operaciones. En la Figura 13.2 se muestra un posible plan de evaluación para la expresión de la Figura 13.1(b). Como ya se ha visto, se pueden usar diferentes algoritmos para cada operación relacional, lo que nos proporciona planes de evaluación alternativos. En la figura se ha elegido la reunión asociativa para una de las operaciones de reunión, mientras que para otra se usa la reunión por mezcla, tras la ordenación de las relaciones por el atributo de la unión, que es *ID*. Donde las líneas se marcan como cauce, la salida del productor se encauza directamente al consumidor sin necesidad de escribir en disco.

Dada una expresión del álgebra relacional, es labor del optimizador de consultas diseñar un plan de evaluación que calcule el mismo resultado que la expresión dada, y de la forma menos costosa a la hora de generar el resultado (o, al menos, no mucho más que la menos costosa).

Para encontrar el plan de evaluación de consultas menos costoso el optimizador necesita generar planes alternativos que produzcan el mismo resultado que la expresión dada y escoger el de menor coste. La generación de planes de evaluación de consultas consta de tres etapas: (1) la generación de expresiones que sean equivalentes lógicamente a la expresión dada, (2) la anotación de las expresiones alternativas resultantes para generar planes distintos de ejecución y (3) la estimación del coste de cada plan de evaluación, y la elección del que se estime que tiene menor coste.

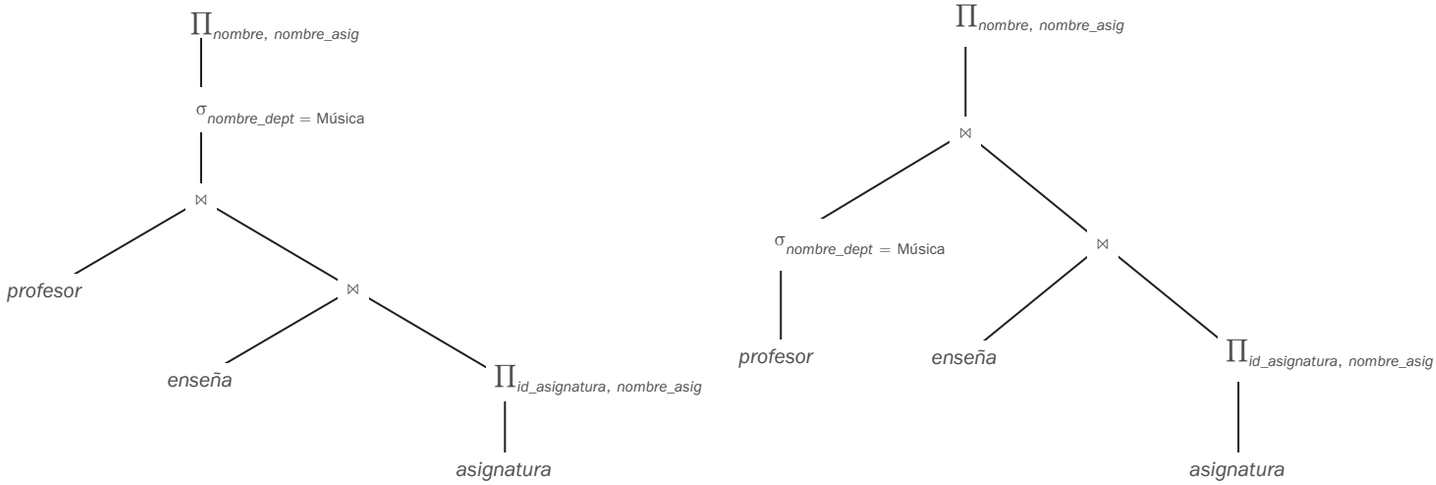
Las etapas (1), (2) y (3) se encuentran entrelazadas en el optimizador de consultas; algunas expresiones se generan y se anotan para dar lugar a planes de evaluación, luego se generan y se anotan otras expresiones, etc. Según se generan planes de evaluación, se estiman sus costes usando información estadística sobre las relaciones, como los tamaños de las relaciones y la profundidad de los índices.

Para implementar la primera etapa el optimizador de consultas debe generar expresiones equivalentes a la expresión dada. Esto se lleva a cabo mediante las *reglas de equivalencia*, que especifican el modo de transformar una expresión en otra lógicamente equivalente. Estas reglas se describen en la Sección 13.2.

En la Sección 13.3 se describe el modo de estimar las estadísticas de los resultados de cada operación en los planes de consultas. El empleo de estas estadísticas con las fórmulas de costes del Capítulo 12 permite estimar los costes de cada operación individual. Los costes individuales se combinan para determinar el coste estimado de la evaluación de la expresión de álgebra relacional dada, como se indicó previamente en la Sección 12.7.

En la Sección 13.4 se describe el modo de escoger un plan de evaluación de consultas. Se puede escoger uno basado en el coste estimado de los planes. Dado que el coste es una estimación, el plan seleccionado no es necesariamente el de menor coste; no obstante, siempre y cuando las estimaciones sean buenas, es probable que el plan sea el de menor coste, o no represente mucho más coste.

Finalmente, las vistas materializadas ayudan a acelerar el procesamiento de ciertas consultas. En la Sección 13.5 se estudia el modo de «mantener» las vistas materializadas, es decir, mantenerlas actualizadas, y la manera de llevar a cabo la optimización de consultas con las vistas materializadas.



(a) Árbol inicial de la expresión

(b) Árbol transformado de la expresión

Figura 13.1. Expresiones equivalentes.

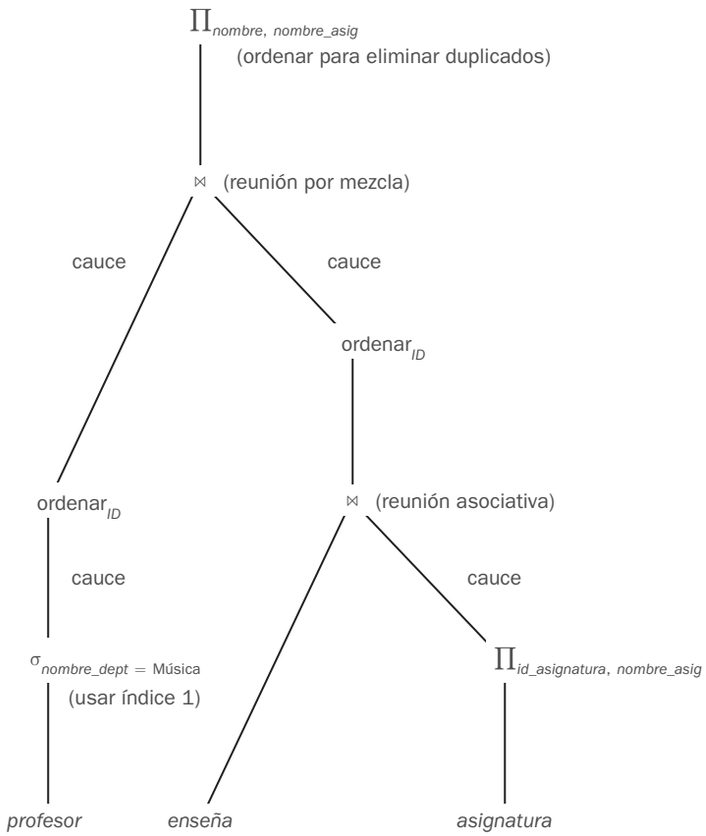


Figura 13.2. Plan de evaluación.

REVISIÓN DE LOS PLANES DE EVALUACIÓN DE CONSULTAS

La mayoría de los sistemas de bases de datos proporcionan algún mecanismo para revisar los planes de evaluación escogidos cuando se ejecuta una consulta. Normalmente es mejor usar la GUI que proporciona el sistema de base de datos para ver los planes de evaluación. Sin embargo, si se usa una interfaz de línea de comandos, muchas bases de datos disponen de un comando «**explain** <consulta>», que muestra el plan de ejecución elegido para la consulta <consulta> especificada. La sintaxis concreta varía de un sistema a otro:

- PostgreSQL usa la sintaxis indicada anteriormente.
- Oracle usa la sintaxis **explain plan for.** Sin embargo, el comando guarda el plan resultante en una tabla llamada «*plan_table*», en lugar de mostrarlo. La consulta «**select** * **from table**(*dbms_xplan.display*);» muestra el plan almacenado.
- DB2 utiliza un mecanismo similar al de Oracle, pero ejecutando el programa db2exfmt para mostrar el plan almacenado.
- SQL Server requiere que se ejecute el comando **set showplan text on** antes de enviar la consulta; después, cuando ya se ha enviado, en lugar de ejecutar la consulta se muestra el plan de evaluación.

Junto con el plan de evaluación también se muestra el coste estimado del plan. Hay que tener en cuenta que el coste no se muestra normalmente en ninguna unidad externa con sentido, como segundos o número de operaciones de E/S, sino en unidades del modelo de coste que use el optimizador. Algunos optimizadores, como el de PostgreSQL, muestran dos números de estimación del coste; el primero indica el coste estimado para generar el primer resultado y el segundo refleja el coste estimado para generar todos los resultados.

13.2. Transformación de expresiones relacionales

Las consultas se pueden expresar de varias maneras diferentes, con costes de evaluación distintos. En este apartado, en lugar de tomar la expresión relacional original, se consideran expresiones alternativas equivalentes.

Se dice que dos expresiones del álgebra relacional son **equivalentes** si, en cada ejemplar legal de la base de datos, las dos expresiones generan el mismo conjunto de tuplas (recuerde que un ejemplar legal de la base de datos es el que satisface todas las restricciones de integridad especificadas en el esquema de la base de datos). Tenga en cuenta que el orden de las tuplas resulta irrelevante; puede que las dos expresiones generen las tuplas en órdenes diferentes, pero se considerarán equivalentes siempre que el conjunto de tuplas sea el mismo.

En SQL las entradas y las salidas son multiconjuntos de tuplas, y se usa la versión para multiconjuntos del álgebra relacional (descrito en el recuadro de la Sección 6.1.4) para evaluar las consultas de SQL. Se dice que dos expresiones de la versión *para multiconjuntos* del álgebra relacional son equivalentes si en cada base de datos legal las dos expresiones generan el mismo multiconjunto de tuplas. El estudio de este capítulo se basa en el álgebra relacional. Las extensiones a la versión para multiconjuntos del álgebra relacional se dejan al lector como ejercicios.

13.2.1. Reglas de equivalencia

Una **regla de equivalencia** establece que dos expresiones son equivalentes. Se puede sustituir la primera por la segunda forma, o viceversa, ya que las dos generan el mismo resultado en cualquier base de datos válida. El optimizador usa las reglas de equivalencia para transformar las expresiones en otras lógicamente equivalentes.

A continuación se enumeran varias reglas generales de equivalencia para las expresiones del álgebra relacional. Algunas de las equivalencias relacionadas aparecen en la Figura 13.3. Se usan θ, θ_1, θ_2, etc., para denotar los predicados, L_1, L_2, L_3, etc., para denotar las listas de atributos y E, E_1, E_2, etc. para denotar las expresiones del álgebra relacional. El nombre de relación r no es más que un caso especial de expresión del álgebra relacional y puede usarse siempre que aparezca E.

1. Las operaciones de selección conjuntivas pueden dividirse en una secuencia de selecciones individuales. Esta transformación se denomina cascada de σ.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. Las operaciones de selección son **conmutativas.**

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. Solo son necesarias las últimas operaciones de una secuencia de operaciones de proyección, las demás pueden omitirse. Esta transformación también puede denominarse cascada de Π.

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{L_n}(E))\ldots)) = \Pi_{L_1}(E)$$

4. Las selecciones pueden combinarse con los productos cartesianos y con las reuniones zeta.

   a. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$

   Esta expresión es precisamente la definición de la reunión zeta.

   b. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

5. Las operaciones de reunión zeta son conmutativas.

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

   Realmente, el orden de los atributos es diferente en el término de la derecha y en el de la izquierda, por lo que la equivalencia no se cumple si se tiene en cuenta el orden de los atributos. Se puede añadir una operación proyección a uno de los lados de la equivalencia para reordenar los atributos de la manera adecuada, pero para mayor sencillez se omite la proyección y se ignora el orden de los atributos en la mayor parte de los ejemplos.

   Recuerde que el operador de reunión natural es simplemente un caso especial del operador de reunión zeta; por tanto, las reuniones naturales también son conmutativas.

6. a. Las operaciones de reunión natural son **asociativas.**

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

   b. Las reuniones zeta son asociativas en el sentido siguiente:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

   donde θ_2 implica solamente atributos de E_2 y de E_3. Cualquiera de estas condiciones puede estar vacía; por tanto, se deduce que la operación producto cartesiano ($\times$) también es asociativa. La conmutatividad y la asociatividad de las operaciones de reunión son importantes para la reordenación de las reuniones en la optimización de las consultas.

7. La operación selección es distributiva por la operación reunión zeta bajo las dos condiciones siguientes:

   a. Cuando todos los atributos de la condición de selección θ_0 implican únicamente los atributos de una de las expresiones (por ejemplo, E_1) que se están reuniendo.

$$\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$$

   b. Cuando la condición de selección θ_1 implica únicamente los atributos de E_1 y θ_2 implica únicamente los atributos de E_2.

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$$

8. La operación proyección es distributiva con la operación reunión zeta bajo las condiciones siguientes:

   a. Sean L_1 y L_2 atributos de E_1 y de E_2, respectivamente. Suponga que la condición de reunión θ implica únicamente los atributos de $L_1 \cup L_2$. Entonces,

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

   b. Considérese una reunión $E_1 \bowtie_\theta E_2$. Sean L_1 y L_2 conjuntos de atributos de E_1 y de E_2, respectivamente. Sean L_3 los atributos de E_1 que están implicados en la condición de reunión θ, pero que no están en $L_1 \cup L_2$, y sean L_4 los atributos de E_2 que están implicados en la condición de reunión θ, pero que no están en $L_1 \cup L_2$. Entonces,

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$

9. Las operaciones de conjuntos unión e intersección son conmutativas.

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

   La diferencia de conjuntos no es conmutativa.

10. La unión y la intersección de conjuntos son asociativas.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11. La operación selección es distributiva con las operaciones de unión, intersección y diferencia de conjuntos.

$$\sigma_P(E_1 - E_2) = \sigma_P(E_1) - \sigma_P(E_2)$$

De manera parecida, la equivalencia anterior, con $-$ sustituido por $\cup$ o por $\cap$, también es válida. Además:

$$\sigma_P(E_1 - E_2) = \sigma_P(E_1) - E_2$$

La equivalencia anterior, con $-$ sustituido por $\cap$, también es válida, pero no se cumple si $-$ se sustituye por $\cup$.

12. La operación proyección es distributiva con respecto a la operación unión.

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$

Esta es solo una lista parcial de las equivalencias. En los ejercicios se estudian más equivalencias que implican a los operadores relacionales extendidos, como la reunión externa y la agregación.

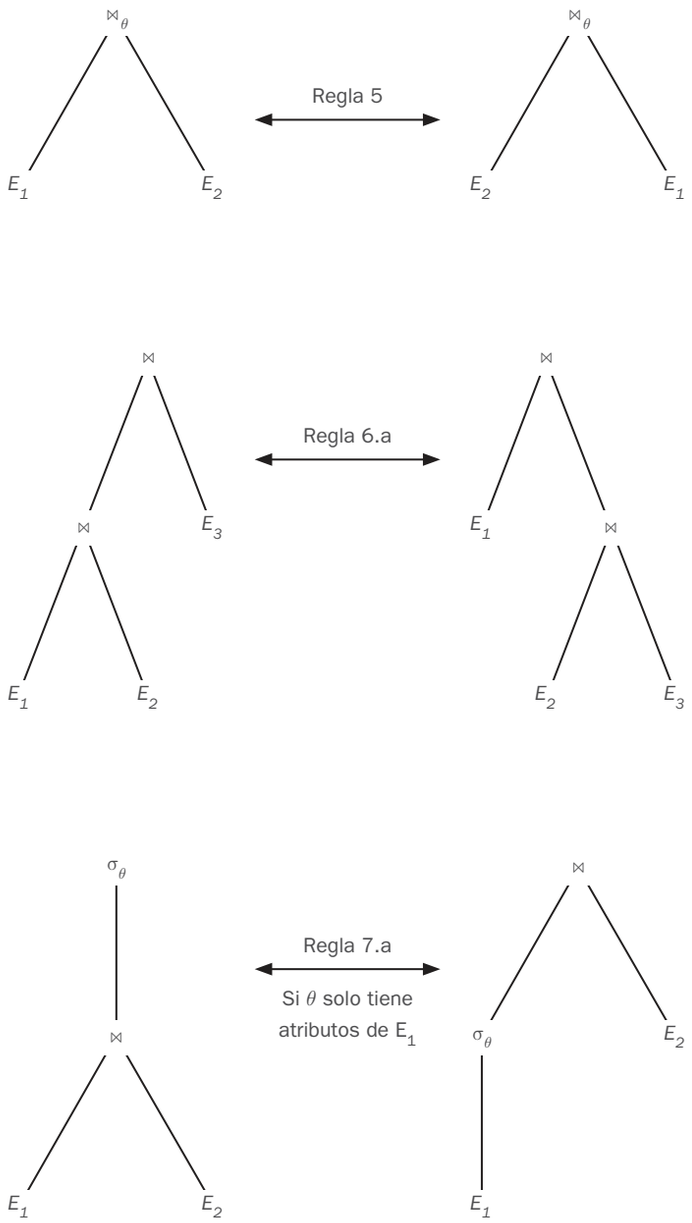


Figura 13.3. Representación gráfica de las equivalencias.

13.2.2. Ejemplos de transformaciones

Ahora se mostrará el empleo de las reglas de equivalencia. Se usará el ejemplo de la universidad con los esquemas de relaciones:

profesor(*ID*, *nombre*, *nombre_dept*, *sueldo*)
enseña(*ID*, *asignatura_id*, *secc_id*, *semestre*, *año*)
asignatura(*asignatura_id*, *nombre_asig*, *nombre_dept*, *créditos*)

En el ejemplo de la Sección 13.1, la expresión:

$$\Pi_{nombre,\ nombre_asig}(\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor \bowtie (enseña \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura)))$$

se transformaba en la expresión siguiente,

$$\Pi_{nombre,\ nombre_asig}((\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor)) \bowtie (enseña \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura)))$$

la cual es equivalente a la expresión algebraica original, pero generando relaciones intermedias de menor tamaño. Esta transformación se puede llevar a cabo empleando la regla 7.a. Recuerde que la regla solo dice que las dos expresiones son equivalentes; no dice que una sea mejor que la otra.

Se pueden usar varias reglas de equivalencia, una tras otra, sobre una consulta o sobre partes de una consulta. Como ejemplo, supóngase que se modifica la consulta original para restringir la atención a los profesores que enseñaron una asignatura en el año 2009. La nueva consulta del álgebra relacional es:

$$\Pi_{nombre,\ nombre_asig}(\sigma_{nombre_dept\ =\ \text{«Música»}\ \wedge\ año\ =\ 2009}(profesor \bowtie (enseña \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura)))$$

No se puede aplicar el predicado de la selección directamente a la relación *profesor*, ya que este implica atributos tanto de la relación *profesor* como de la relación *enseña*. No obstante, se puede aplicar antes la regla 6.a (asociatividad de la reunión natural) para transformar la reunión:

$$profesor \bowtie (enseña \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura))$$

en:

$$(profesor \bowtie enseña) \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura):$$

$$\Pi_{nombre,\ nombre_asig}(\sigma_{nombre_dept\ =\ \text{«Música»}\ \wedge\ año\ =\ 2009}((profesor \bowtie enseña) \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura)))$$

Luego, empleando la regla 7.a, se puede reescribir la consulta como:

$$\Pi_{nombre,\ nombre_asig}((\sigma_{nombre_dept\ =\ \text{«Música»}\ \wedge\ año\ =\ 2009}(profesor \bowtie enseña)) \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura))$$

Examinemos la subexpresión de selección de esta expresión. Empleando la regla 1 se puede dividir la selección en dos, para obtener la subexpresión siguiente:

$$\sigma_{nombre_dept\ =\ \text{«Música»}}(\sigma_{año\ =\ 2009}(profesor \bowtie enseña))$$

Las dos expresiones anteriores seleccionan tuplas con *nombre_dept* = «Música» y *año* = 2009. Sin embargo, la última forma de la expresión ofrece una nueva oportunidad de aplicar la regla 7.a («llevar a cabo primero las selecciones»), que da lugar a la subexpresión:

$$\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor) \bowtie \sigma_{año\ =\ 2009}(enseña)$$

La Figura 13.4 muestra la expresión inicial y la expresión final después de todas estas transformaciones. También se podría haber usado la regla 7.b para obtener directamente la expresión final, sin usar la regla 1 para dividir la selección en dos selecciones. De hecho, la regla 7.b puede obtenerse de las reglas 1 y 7.a.

Se dice que un conjunto de reglas de equivalencia es **mínimo** si no se puede obtener ninguna regla a partir de una reunión de las demás. El ejemplo anterior muestra que el conjunto de reglas de equivalencia de la Sección 13.2.1 no es mínimo. Se puede generar una expresión equivalente a la original de diferentes maneras; el número de formas diversas de generar una expresión aumenta cuando se usa un conjunto de reglas de equivalencia que no es mínimo. Los optimizadores de consultas, por tanto, usan conjuntos mínimos de reglas de equivalencia.

Considere ahora la siguiente forma de la consulta de ejemplo:

$$\Pi_{nombre,\ nombre_asig}((\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor) \bowtie enseña) \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura))$$

Cuando se calcula la subexpresión:

$$(\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor) \bowtie enseña)$$

se obtiene una relación cuyo esquema es:

$$(ID, nombre, nombre_dept, sueldo, asignatura_id, secc_id, semestre, año)$$

Es posible eliminar varios atributos del esquema forzando las proyecciones de acuerdo con las reglas de equivalencia 8.a y 8.b. Los únicos atributos que se deben conservar son los que aparecen en el resultado de la consulta y los que se necesitan para procesar las operaciones subsiguientes. Al eliminar los atributos innecesarios, se reduce el número de columnas del resultado intermedio. Por tanto, se reduce el tamaño del resultado intermedio. En el ejemplo, los únicos atributos que se necesitan de la reunión de *profesor* y de *enseña* son *nombre* y *asignatura_id*. Por tanto, se puede modificar la expresión hasta:

$$\Pi_{nombre,\ nombre_asig}((\Pi_{nombre,\ asignatura_id}((\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor) \bowtie enseña) \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura))$$

La proyección:

$$\Pi_{nombre,\ asignatura_id}$$

reduce el tamaño de los resultados intermedios de la reunión.

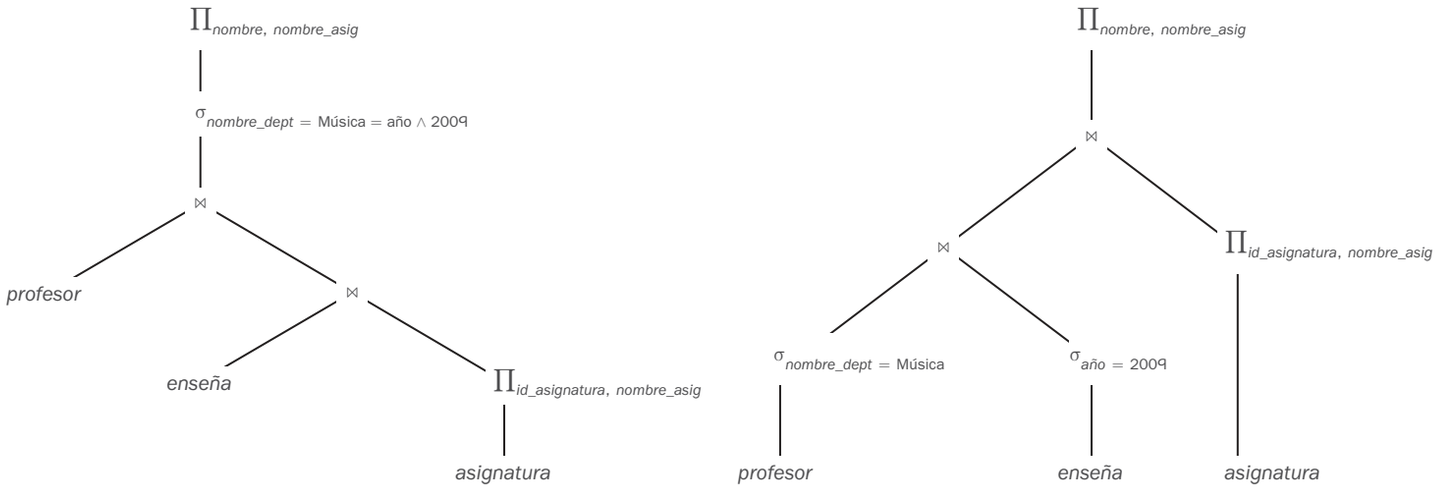


Figura 13.4. Múltiples transformaciones.

13.2.3. Ordenación de las reuniones

Es importante conseguir una buena ordenación de las operaciones reunión para reducir el tamaño de los resultados temporales; por tanto, la mayor parte de los optimizadores de consultas prestan mucha atención al orden de las reuniones. Como se mencionó en el Capítulo 6 y en la regla de equivalencia 6.a, la operación reunión natural es asociativa. Por tanto, para todas las relaciones r_1, r_2 y r_3,

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

Aunque estas expresiones sean equivalentes, los costes de calcular cada una de ellas pueden ser diferentes. Considere una vez más la expresión:

$$\Pi_{nombre,\ nombre_asig}((\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor)) \bowtie enseña \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura))$$

Se podría escoger calcular primero:

$$enseña \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura)$$

y luego combinar el resultado con:

$$\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor)$$

Sin embargo, es probable que:

$$enseña \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura)$$

sea una relación de gran tamaño, ya que contiene una tupla por cada asignatura que se enseña. En cambio,

$$\sigma_{nombre_dept\ =\ \text{«Música»}}(profesor)) \bowtie enseña$$

probablemente sea una relación de tamaño pequeño. Para comprobar que es así, se observa que, dado que la universidad tiene un gran número de departamentos, es probable que solo un pequeño grupo de profesores estén asociados al departamento de Música. Por tanto, la expresión anterior da lugar a una tupla por cada asignatura que enseña un profesor del departamento de Música. Así, la relación temporal que se debe almacenar es menor que si se hubiera calculado primero:

$$enseña \bowtie \Pi_{asignatura_id,\ nombre_asig}(asignatura).$$

Hay otras opciones a considerar a la hora de evaluar la consulta. No hay que preocuparse del orden en que aparecen los atributos en las reuniones, ya que resulta sencillo cambiarlo antes de mostrar el resultado. Por tanto, para todas las relaciones r_1 y r_2,

$$r_1 \bowtie r_2 = r_2 \bowtie r_1$$

Es decir, la reunión natural es conmutativa (regla de equivalencia 5).

Mediante la asociatividad y la conmutatividad de la reunión natural (reglas 5 y 6) considere la siguiente expresión del álgebra relacional:

$$(profesor \bowtie \Pi_{asignatura_id,\ nombre_asig}\ (asignatura)) \bowtie enseña$$

Observe que no existen atributos en común entre:

$$\Pi_{asignatura_id,\ nombre_asig}\ (asignatura)$$

y *profesor*, por lo que la reunión no es más que un producto cartesiano. Si hay a tuplas en *profesor* y b tuplas en:

$$\Pi_{asignatura_id,\ nombre_asig}\ (asignatura),$$

este producto cartesiano genera $a * b$ tuplas, una por cada par posible de tuplas de profesor y de asignatura (independientemente de si el *profesor* enseñó la asignatura). Este producto cartesiano puede generar una relación temporal muy grande. Sin embargo, si el usuario ha introducido la expresión anterior se puede utilizar la asociatividad y conmutatividad de la reunión natural para transformar la expresión en la más eficiente:

$$(profesor \bowtie enseña) \bowtie \Pi_{asignatura_id,\ nombre_asig}\ (asignatura)$$

13.2.4. Enumeración de expresiones equivalentes

Los optimizadores de consultas usan las reglas de equivalencia para generar de manera sistemática expresiones equivalentes a la expresión de consulta dada. Conceptualmente, el proceso se desarrolla como se describe en la Figura 13.5. Dada una expresión E, el conjunto de expresiones equivalentes EQ inicialmente solo contiene a E. A continuación, cada expresión de EQ se ve si coincide con alguna regla de equivalencia. Si una expresión, por ejemplo E_i de alguna subexpresión e_i de E_i (que en algún caso especial puede ser el propio E_i), coincide con uno de los lados de la regla de equivalencia, el optimizador genera una nueva expresión en la que se transforma e_i para que se convierta en el otro lado de la regla. La expresión resultante se añade a EQ. Este proceso continúa hasta que no se puedan generar expresiones nuevas.

El proceso anterior resulta costoso tanto en espacio como en tiempo, pero los optimizadores pueden reducir drásticamente el espacio y el tiempo aplicando dos ideas clave:

1. Si se genera una expresión E' de una expresión E_1 usando una regla de equivalencia sobre una subexpresión e_i, entonces E' y E_1 tienen subexpresiones idénticas excepto para e_i y su transformación. Aunque e_i y su versión transformada suelen compartir muchas subexpresiones idénticas. Las técnicas de representación de expresiones que permiten que ambas expresiones puedan apuntar a subexpresiones compartidas pueden reducir significativamente el espacio requerido.
2. No es necesario generar siempre todas las expresiones que se puedan generar con las reglas de equivalencia. Si un optimizador tiene en cuenta la estimación de coste de la evaluación, puede ser capaz de examinar algunas de las expresiones, tal como se vio en la Sección 13.4. Se puede reducir el tiempo de optimización necesario utilizando dichas técnicas.

   Estos temas se volverán a tratar en la Sección 13.4.2.

```
procedure generaTodasEquivalentes(E)
EQ = {E}
repeat
   Casa cada expresión E_i en EQ con cada regla de equivalencia R_j
   if alguna subexpresión e_i de E_i casa una parte de R_j
   Crea una nueva expresión E´ que es idéntica a E_i, excepto que e_i
      se transforma para casar con la otra parte de R_j
   Añade E´ a EQ si no está todavía en EQ
until no se puedan añadir nuevas expresiones a EQ
```

Figura 13.5. Procedimiento para generar todas las expresiones equivalentes.

13.3. Estimación de las estadísticas de los resultados de las expresiones

El coste de cada operación depende del tamaño y de otras estadísticas de sus valores de entrada. Dada una expresión como $a \bowtie (b \bowtie c)$, para estimar el coste de combinar a con $(b \bowtie c)$ hay que hacer estimaciones de estadísticas como el tamaño de $b \bowtie c$.

En esta sección se relacionarán en primer lugar algunas estadísticas de las relaciones de bases de datos que se almacenan en los catálogos de los sistemas de bases de datos y luego se mostrará el modo de usar las estadísticas para estimar estadísticas de los resultados de varias operaciones relacionales.

Algo que quedará claro más adelante en este apartado es que las estimaciones no son muy precisas, ya que se basan en suposiciones que pueden no cumplirse exactamente. El plan de evaluación de consultas que tenga el coste estimado de ejecución más reducido puede, por tanto, no tener el coste real de ejecución más bajo. Sin embargo, la experiencia real demuestra que, aunque las estimaciones no sean muy precisas, los planes con los costes estimados más reducidos tienen costes de ejecución reales que, o bien son los más reducidos, o bien se hallan cercanos al menor coste real de ejecución.

13.3.1. Información de catálogo

Los catálogos de los sistemas de bases de datos almacenan la siguiente información estadística sobre las relaciones de las bases de datos:

- n_r, el número de tuplas de la relación r.
- b_r, el número de bloques que contienen tuplas de la relación r.
- l_r, el tamaño de cada tupla de la relación r en bytes.
- f_r, el factor de bloques de la relación r —el factor de bloqueo de la relación r que caben en un bloque.
- $V(A, r)$, el número de valores distintos que aparecen en la relación r para el atributo A. Este valor es igual que el tamaño de $\Pi_A(r)$. Si A es una clave de la relación r, $V(A, r)$ es n_r.

La última estadística, $V(A, r)$, también puede calcularse para conjuntos de atributos, si se desea, en vez de solo para atributos aislados. Por tanto, dado un conjunto de atributos, A, $V(A, r)$ es el tamaño de $\Pi_A(r)$.

Si se supone que las tuplas de la relación r se almacenan físicamente juntas en un archivo, se cumple la ecuación siguiente:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

Las estadísticas sobre los índices, como las alturas de los árboles B^+ y el número de páginas hojas de los índices, también se conservan en el catálogo.

Si se desean conservar estadísticas precisas, cada vez que se modifica una relación también hay que actualizar las estadísticas. Esta actualización supone una sobrecarga sustancial. Por tanto, la mayor parte de los sistemas no actualizan las estadísticas con cada modificación. En lugar de eso, actualizan las estadísticas durante los periodos de poca carga del sistema. En consecuencia, puede que las estadísticas usadas para escoger una estrategia de procesamiento de consultas no sean completamente exactas. Sin embargo, si no se producen demasiadas actualizaciones en los intervalos entre las actualizaciones de las estadísticas, estas serán lo bastante precisas como para proporcionar una buena estimación de los costes relativos de los diferentes planes.

La información estadística indicada aquí está simplificada. Los optimizadores reales suelen conservar información estadística adicional para mejorar la precisión de sus estimaciones de costes de los planes de evaluación. Por ejemplo, la mayoría de las bases de datos almacenan la distribución de los valores de cada atributo en forma de **histograma:** en los histogramas, los valores del atributo se dividen en una serie de intervalos, y con cada intervalo el histograma asocia el número de tuplas cuyo valor del atributo se halla en ese intervalo. En la Figura 13.6 se muestra un ejemplo de histograma para un atributo entero que toma valores en el intervalo de 1 a 25.

Los histogramas usados en los sistemas de bases de datos habitualmente registran el número de valores distintos en cada intervalo, además del número de tuplas en ese intervalo.

Como ejemplo de histograma, el rango de valores del atributo *edad* de la relación *persona* puede dividirse en 0 – 9, 10 – 19, …, 90 – 99 (suponiendo una edad máxima de 99). Con cada rango se almacena un recuento del número de tuplas *persona* cuyos valores de *edad* se hallan en ese rango, junto con el número de valores distintos de edad que se encuentran en ese intervalo. Sin la información del histograma, un optimizador tendría que suponer que la distribución de los valores es uniforme; es decir, que cada intervalo tiene el mismo recuento.

Un histograma ocupa muy poco espacio, por lo que los histogramas sobre varios atributos diferentes se pueden almacenar en el catálogo del sistema. En los sistemas de bases de datos se usan varios tipos de histogramas. Por ejemplo, el **histograma de equianchura** divide los valores en intervalos de igual tamaño, mientras que el histograma de **equiprofundidad** ajusta las fronteras de los intervalos de forma que cada intervalo tenga el mismo número de valores.

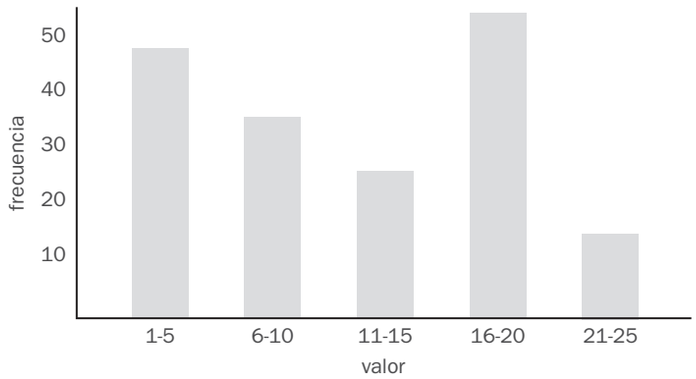


Figura 13.6. Ejemplo de histograma.

13.3.2. Estimación de tamaño de la selección

La estimación del tamaño del resultado de una operación selección depende del predicado de la selección. En primer lugar se considerará un solo predicado de igualdad, luego un solo predicado de comparación y, finalmente, combinaciones de predicados.

- $\sigma_{A = a}(r)$: si se supone una distribución uniforme de los valores (es decir, que cada valor aparece con la misma probabilidad), se puede estimar que el resultado de la selección tiene $n_r/V(A, r)$ tuplas, suponiendo que el valor a aparece en el atributo A de algún registro de r. La suposición de que el valor a de la selección aparece en algún registro suele ser cierta, y las estimaciones de costes suelen hacerla de manera implícita. No obstante, no suele ser realista suponer que cada valor aparece con la misma probabilidad. El atributo *asignatura_id* de la relación *enseña* es un ejemplo en el que esta suposición no es válida. Es razonable esperar que un curso de grado que sea popular tenga muchos más estudiantes que un curso especializado de posgrado. Por tanto, ciertos valores de *asignatura_id* aparecen con mayor probabilidad que otros. Pese al hecho de que la suposición de distribución uniforme no suele ser correcta, en muchos casos resulta una aproximación razonable a la realidad, y ayuda a mantener una presentación relativamente sencilla.

  Si se dispone de un histograma sobre el atributo A, se puede localizar el intervalo que contiene el valor a, y modificar la estimación anterior $n_r/V(A, r)$ usando el contador de frecuencia para el rango en lugar de n_r, y el número de valores distintos que aparecen en el intervalo en lugar de $V(A, r)$.
- $\sigma_{A \leq v}(r)$: considere una selección de la forma $\sigma_{A \leq v}(r)$. Si el valor real usado en la comparación (v) está disponible en el momento de la estimación del coste, puede hacerse una estimación más precisa.

  Los valores mínimos y máximos (min(A, r) y max(A, r)) del atributo pueden almacenarse en el catálogo. Suponiendo que los valores están distribuidos de manera uniforme, se puede estimar el número de registros que cumplirán la condición $A \leq v$ como 0 si $v <$ mín(A, r), como n_r si $v \geq$ máx(A, r), y:

  $$nr \cdot \frac{v - \min(A, r)}{\max(A, r) - \min(A, r)}$$

  en otro caso.

  Si hay disponible un histograma sobre el atributo A, se puede obtener una estimación más precisa; se dejan los detalles como ejercicio para el lector. En algunos casos, como cuando la consulta forma parte de un procedimiento almacenado, puede que el valor v no esté disponible cuando se optimice la consulta. En esos casos, se supone que aproximadamente la mitad de los registros cumplen la condición de comparación. Es decir, se supone que el resultado tiene $n_r/2$ tuplas; la estimación puede resultar muy imprecisa, pero es lo mejor que se puede hacer sin más información.
- Selecciones complejas:
  - **Conjunción:** una *selección conjuntiva* es una selección de la forma:

    $$\sigma_{\theta_1 \wedge \theta_2 \wedge \ldots \wedge \theta_n}(r)$$

    Se puede estimar el tamaño del resultado de esta selección: para cada θ_i, se estima el tamaño de la selección $\sigma_{\theta_i}(r)$, denotada por s_i como se ha descrito anteriormente. Por tanto, la probabilidad de que una tupla de la relación satisfaga la condición de selección θ_i es s_i/n_r.

    La probabilidad anterior se denomina **selectividad** de la selección $\sigma_{\theta i}(r)$. Suponiendo que las condiciones sean *independientes* entre sí, la probabilidad de que una tupla sa-

tisfaga todas las condiciones es simplemente el producto de todas estas probabilidades. Por tanto, se estima el número de tuplas de la selección completa como:

$$n_r * \frac{s_1 * s_2 * \ldots * s_n}{n_r^n}$$

– **Disyunción:** una *selección disyuntiva* es una selección de la forma:

$$\sigma_{\theta_1 \vee \theta_2 \vee \ldots \vee \theta_n}(r)$$

Una condición disyuntiva se satisface por la unión de todos los registros que satisfacen las condiciones simples θ_i.

Como anteriormente, sea s_i/n_r la probabilidad de que una tupla satisfaga la condición θ_i. La probabilidad de que la tupla satisfaga la disyunción es, pues, 1 menos la probabilidad de que no satisfaga *ninguna* de las condiciones:

$$1 - \left(1 - \frac{s_1}{n_r}\right) * \left(1 - \frac{s_2}{n_r}\right) * \ldots * \left(1 - \frac{s_n}{n_r}\right)$$

Multiplicando este valor por n_r se obtiene el número estimado de tuplas que satisfacen la selección.

– **Negación:** a falta de valores nulos, el resultado de una selección $\sigma_{\neg\theta}(r)$ es simplemente las tuplas de r que no están en $\sigma_\theta(r)$. Ya sabemos el modo de estimar el número de tuplas de $\sigma_\theta(r)$. El número de tuplas de $\sigma_{\neg\theta}(r)$ se estima, por tanto, que es $n(r)$ menos el número estimado de tuplas de $\sigma_\theta(r)$.

Se pueden tener en cuenta los valores nulos estimando el número de tuplas para las que la condición θ se evalúa como *desconocida*, y restar ese número de la estimación anterior que ignora los valores nulos. La estimación de ese número exige conservar estadísticas adicionales en el catálogo.

13.3.3. Estimación del tamaño de las reuniones

En esta sección se verá el modo de estimar el tamaño del resultado de una reunión.

El producto cartesiano $r \times s$ contiene $n_r * n_s$ tuplas. Cada tupla de $r \times s$ ocupa $t_r + t_s$ bytes, de donde se puede calcular el tamaño del producto cartesiano.

La estimación del *tamaño* de una reunión natural resulta algo más complicada que la estimación del tamaño de una selección del producto cartesiano. Sean $r(R)$ y $s(S)$ dos relaciones.

- Si $R \cap S = \varnothing$, es decir, las relaciones no tienen ningún atributo en común, entonces $r \bowtie s$ es igual que $r \times s$, y se puede usar la técnica de estimación anterior para los productos cartesianos.
- Si $R \cap S$ es una clave para R, entonces se sabe que cada tupla de s se combinará como máximo con una tupla de r. Por tanto, el número de tuplas de $r \bowtie s$ no es mayor que el número de tuplas de s. En el caso de que $R \cap S$ sea clave de S, es simétrico al caso que se acaba de describir. Si $R \cap S$ forma una clave externa de S, que hace referencia a R, el número de tuplas de $r \bowtie s$ es exactamente el mismo que el número de tuplas de s.
- El caso más difícil es que $R \cap S$ no sea clave de R ni de S. En ese caso se supone, como se hizo para las selecciones, que todos los valores aparecen con la misma probabilidad. Considere una tupla t de r y suponga que $R \cap S = \{A\}$. Se estima que la tupla t produce:

$$\frac{n_s}{V(A,s)}$$

tuplas en $r \bowtie s$, ya que este número es el número promedio de tuplas de s con un valor dado para los atributos A. Considerando todas las tuplas de r, se estima que hay:

$$\frac{n_s * n_s}{V(A,s)}$$

tuplas en $r \bowtie s$. Observe que, si se invierten los papeles de r y de s en la estimación anterior, se obtiene una estimación de:

$$\frac{n_s * n_s}{V(A,r)}$$

tuplas en $r \bowtie s$. Estas dos estimaciones son diferentes si $V(A, r) \neq V(A, s)$. Si esta situación se produce, probablemente haya tuplas pendientes que no participen en la reunión. Por tanto, es probable que la menor de las dos estimaciones sea la más precisa.

La estimación anterior del tamaño de la reunión puede ser demasiado elevada si los valores de $V(A, r)$ para el atributo A de r tienen pocos valores en común con los valores de $V(A, s)$ para el atributo A de s. No obstante, es improbable que se dé esta situación en la realidad, ya que las tuplas pendientes o bien no existen o solo constituyen una pequeña fracción de las tuplas, en la mayor parte de las relaciones reales.

Lo más importante es que la estimación anterior depende de la suposición de que todos los valores aparecen con la misma probabilidad. Si esta suposición no resulta válida hay que usar técnicas más sofisticadas para la estimación del tamaño. Por ejemplo, si se tuviesen histogramas sobre los atributos de la reunión de ambas relaciones, y ambos histogramas tuviesen los mismos intervalos, se podría usar la técnica de la estimación anterior en los dos intervalos usando el número de filas con valores en el intervalo en lugar de n_r o de n_s, y el número de valores distintos en el intervalo en lugar de $V(A, r)$ o de $V(A, s)$. Después se sumarían las estimaciones de tamaño obtenidas para cada intervalo para calcular el tamaño total. Como ejercicio, se deja al lector el caso en que ambas relaciones tengan histogramas sobre el atributo de la reunión, pero con intervalos diferentes en los histogramas.

Se puede estimar el tamaño de una reunión zeta $r \bowtie_\theta s$ reescribiendo la reunión como $\sigma_\theta(r \times s)$ y empleando las estimaciones de tamaño de los productos cartesianos junto con las estimaciones de tamaño de las selecciones, que se vieron en la Sección 13.3.2.

Para ilustrar todas estas formas de estimar el tamaño de las reuniones, considérese la expresión:

$$estudiante \bowtie enseña$$

Supóngase que se dispone de la siguiente información de catálogo sobre las dos relaciones:

- $n_{estudiante} = 5.000$.
- $f_{estudiante} = 50$, lo que implica que $b_{estudiante} = 5.000/50 = 100$.
- $n_{enseña} = 10.000$.
- $f_{enseña} = 25$, lo que implica que $b_{enseña} = 10.000/25 = 400$.
- $V(ID, enseña) = 2.500$, lo que implica que solo la mitad de los estudiantes ha terminado la asignatura (esto no es realista, pero se utiliza para demostrar que las estimaciones son correctas incluso en este caso) y en promedio, cada estudiante que haya cursado la asignatura se ha matriculado en cuatro asignaturas.

El atributo *ID* en *enseña* es una clave externa de *estudiante*, y no habrá valores nulos en *enseña.ID*, ya que *ID* forma parte de la clave primaria de enseña, por tanto la relación $estudiante \bowtie enseña$ es exactamente $n_{enseña}$, que es 10.000.

Calculemos ahora las estimación de tamaño de $estudiante \bowtie enseña$ sin usar la información sobre las claves externas. Como $V(ID, enseña) = 2.500$ y $V(ID, estudiante) = 10.000$, las dos estimaciones que se obtienen son $5.000 * 10.000/2.500 = 20.000$ y $5.000 * 10.000/10.000 = 5.000$, y se escoge la menor. En este caso, la menor de las estimaciones es igual que la que se calculó anteriormente a partir de la información sobre las claves externas.

CÁLCULO Y MANTENIMIENTO DE LAS ESTADÍSTICAS

Conceptualmente, las estadísticas sobre relaciones se pueden ver como vistas materializadas que se deberían mantener automáticamente cuando se modifican las relaciones. Desafortunadamente, el mantenimiento actualizado de las estadísticas tras cada inserción, borrado o actualización puede ser muy costoso. Por otra parte, los optimizadores no necesitan estadísticas exactas: un error en un pequeño porcentaje puede hacer que se elija un plan que no sea óptimo, pero el plan alternativo elegido probablemente tenga un coste que es un pequeño porcentaje peor que el coste óptimo. Por tanto, resulta aceptable disponer de estadísticas que sean aproximadas.

Los sistemas de bases de datos reducen el coste de generar y mantener las estadísticas, como se indica a continuación, explotando el hecho de que sean aproximadas.

- Las estadísticas suelen calcularse de una muestra de los datos subyacentes, en lugar de examinar la colección completa de datos. Por ejemplo, se puede calcular un histograma bastante aproximado de una muestra de unos cientos de tuplas, incluso de una relación con millones de ellas. Sin embargo, la muestra debe ser una **muestra aleatoria;** una muestra que no sea aleatoria puede tener una representación excesiva de una parte de la relación y proporcionar resultados sesgados. Por ejemplo, si se usa una muestra de los profesores para calcular un histograma de sueldos, si esta tiene una sobrerrepresentación de los profesores de menores sueldos el histograma generará estimaciones incorrectas. Los sistemas de bases de datos actuales realizan muestreos aleatorios para generar las estadísticas. Consulte las notas bibliográficas para más referencias sobre el muestreo.
- Las estadísticas no se mantienen tras cada actualización de la base de datos. De hecho, algunos sistemas de base de datos nunca actualizan las estadísticas automáticamente. Confían en que los administradores ejecuten periódicamente un comando de actualización de estadísticas. Oracle y PostgreSQL proporcionan el comando de SQL **analyze,** que genera estadísticas de las relaciones que se indiquen, o de todas las relaciones. IBM DB2 dispone de un comando equivalente llamado **runstats.** Consulte los manuales para más información. Debería entender que los optimizadores a veces eligen malos planes debido a estadísticas incorrectas. Muchos sistemas de bases de datos como IBM DB2, Oracle y SQL Server actualizan las estadísticas automáticamente en ciertos momentos. Por ejemplo, el sistema puede mantener un registro aproximado de cuántas tuplas existen en una relación y recalcular las estadísticas si este número cambia significativamente. Otra aproximación consiste en comparar la cardinalidad estimada de una relación con la real cuando se ejecuta una consulta, y si difieren significativamente, inicia una actualización de las estadísticas para esa relación.

13.3.4. Estimación del tamaño de otras operaciones

A continuación se esbozará el modo de estimar el tamaño de los resultados de otras operaciones del álgebra relacional.

- **Proyección:** el tamaño estimado (número de registros o número de tuplas) de una proyección de la forma $\Pi_A(r)$ es $V(A, r)$, ya que la proyección elimina los duplicados.
- **Agregación:** el tamaño de ${}_A\mathcal{G}_F(r)$ es simplemente $V(A, r)$, ya que hay una tupla de ${}_A\mathcal{G}_F(r)$ por cada valor distinto de A.
- **Operaciones de conjuntos:** si las dos entradas de una operación de conjuntos son selecciones de la misma relación se puede reescribir la operación de conjuntos como disyunciones, conjunciones o negaciones. Por ejemplo, $\sigma_{\theta_1}(r) \cup \sigma_{\theta_2}(r)$ se puede reescribir como $\sigma_{\theta_1 \vee \theta_2}(r)$. De manera parecida, las intersecciones se pueden reescribir como conjunciones y la diferencia de conjuntos empleando la negación, siempre que las dos relaciones que participan en la operación de conjuntos sean selecciones de la misma relación. Luego se pueden usar las estimaciones de las selecciones que impliquen conjunciones, disyunciones y negaciones de la Sección 13.3.2.

  Si las entradas no son selecciones de la misma relación, los tamaños se estiman de esta manera: el tamaño estimado de $r \cup s$ es la suma de los tamaños de r y de s. El tamaño estimado de $r \cap s$ es el mínimo de los tamaños de r y de s. El tamaño estimado de $r - s$ es el mismo tamaño de r. Las tres estimaciones pueden ser imprecisas, pero proporcionan cotas superiores para los tamaños.
- **Reunión externa:** el tamaño estimado de $r ⟕ s$ es el tamaño de $r \bowtie s$ más el tamaño de r; el de $r ⟖ s$ es simétrico, mientras que el de $r ⟗ s$ es el tamaño de $r \bowtie s$ más los tamaños de r y s. Las tres estimaciones pueden ser imprecisas, pero proporcionan cotas superiores para los tamaños.

13.3.5. Estimación del número de valores distintos

Para las selecciones, el número de valores distintos de un atributo (o de un conjunto de atributos) A en el resultado de una selección, $V(A, \sigma_\theta(r))$, puede estimarse de la manera siguiente:

- Si la condición de selección θ obliga a que A adopte un valor especificado (por ejemplo, $A = 3$), $V(A, \sigma_\theta(r)) = 1$.
- Si θ obliga a que A adopte un valor de entre un conjunto especificado de valores (por ejemplo, $(A = 1 \vee A = 3 \vee A = 4)$), entonces $V(A, \sigma_\theta(r))$ se define como el número de valores especificados.
- Si la condición de selección θ es de la forma $A\ op\ v$, donde op es un operador de comparación, $V(A, \sigma_\theta(r))$ se estima que es $V(A, r) * s$, donde s es la selectividad de la selección.
- En todos los demás casos de selecciones se da por supuesto que la distribución de los valores de A es independiente de la distribución de los valores para los que se especifican las condiciones de selección y se usa una estimación aproximada de $\min(V(A, r), n_{\sigma_\theta(r)})$. Se puede obtener una estimación más precisa para este caso usando la teoría de la probabilidad, pero la aproximación anterior funciona bastante bien.

Para las reuniones, el número de valores distintos de un atributo (o de un conjunto de atributos) A en el resultado de una reunión, $V(A, r \bowtie s)$, puede estimarse de la manera siguiente:

- Si todos los atributos de A proceden de r, $V(A, r \bowtie s)$ se estima como $\min(V(A, r), n_{r \bowtie s})$, y de manera parecida, si todos los atributos de A proceden de s, $V(A, r \bowtie s)$ se estima que es $\min(V(A, s), n_{r \bowtie s})$.
- Si A contiene atributos $A1$ de r y $A2$ de s, entonces $V(A, r \bowtie s)$ se estima como:

$$\min(V(A1, r) * V(A2-A1, s), V(A1-A2, r) * V(A2, s), n_{r \bowtie s})$$

  Observe que algunos atributos pueden estar en $A1$ y en $A2$, y que $A1-A2$ y $A2-A1$ denotan, respectivamente, a los atributos de A que solo proceden de r y a los atributos de A que solo proceden de s. Nuevamente, se pueden obtener estimaciones más precisas usando la teoría de la probabilidad, pero las aproximaciones anteriores funcionan bastante bien.

Las estimaciones de los distintos valores son directas para las proyecciones: son iguales en $\Pi_A(r)$ que en r. Lo mismo resulta válido para los atributos de agrupación de las agregaciones. Para los resultados de **sum** (suma), **count** (cuenta) y **average** (promedio), se puede suponer, por simplificar, que todos los valores agregados son distintos. Para **min**(A) y **max**(A), el número de valores distintos puede estimarse como $\min(V(A, r), V(G, r))$, donde G denota los atributos de agrupamiento. Se omiten los detalles de la estimación de los valores distintos para otras operaciones.

13.4. Elección de los planes de evaluación

La generación de expresiones es solo una parte del proceso de optimización de consultas, ya que cada operación de la expresión puede implementarse con algoritmos diferentes. Un plan de evaluación define exactamente el algoritmo que se usará para cada operación y el modo en que se coordinará la ejecución de las operaciones.

Dado un plan de evaluación, se puede estimar su coste empleando las estadísticas estimadas mediante las técnicas de la Sección 13.3 junto con las estimaciones de costes de varios algoritmos y métodos de evaluación descritos en el Capítulo 12.

Un **optimizador basado en el coste** explora el espacio de todos los planes de evaluación de consultas equivalentes a la consulta dada y selecciona la que tenga el menor coste estimado. Ya se ha visto cómo se pueden usar las reglas de equivalencia para generar planes equivalentes. Sin embargo, la optimización basada en coste con reglas de equivalencia arbitrarias es muy complicada. Primero se va a tratar una versión simple de la optimización basada en coste, que implica solo la selección de algoritmos de reunión y ordenar reunión, en la Sección 13.4.1. Después, en la Sección 13.4.2, se trata cómo se puede construir un optimizador de propósito general basado en reglas de equivalencia, sin entrar en detalles.

Explorar el espacio de todos los planes posibles puede ser muy costoso para consultas complejas. La mayoría de los optimizadores incluyen heurísticas que reducen el coste de la optimización de consultas, con el riesgo de no encontrar el plan óptimo. Se estudian las heurísticas en la Sección 13.4.3.

13.4.1. Selección del orden de reunión basada en coste

La consulta más común en SQL consiste en una reunión de unas pocas relaciones, con un predicado de reunión y selecciones indicadas en la cláusula **where.** En esta sección se va a considerar el problema de elegir el orden de reunión óptimo para este tipo de consulta.

Para las consultas de reunión complejas, el número de planes de consulta diferentes que son equivalentes a un plan dado puede ser grande. A modo de ejemplo, considere la expresión:

$$r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$$

en que las reuniones se expresan sin ninguna ordenación. Con $n = 3$, hay 12 ordenaciones diferentes:

$r_1 \bowtie (r_2 \bowtie r_3)$	$r_1 \bowtie (r_3 \bowtie r_2)$	$(r_2 \bowtie r_3) \bowtie r_1$	$(r_3 \bowtie r_2) \bowtie r_1$
$r_2 \bowtie (r_1 \bowtie r_3)$	$r_2 \bowtie (r_3 \bowtie r_1)$	$(r_1 \bowtie r_3) \bowtie r_2$	$(r_3 \bowtie r_1) \bowtie r_2$
$r_3 \bowtie (r_1 \bowtie r_2)$	$r_3 \bowtie (r_2 \bowtie r_1)$	$(r_1 \bowtie r_2) \bowtie r_3$	$(r_2 \bowtie r_1) \bowtie r_3$

En general, con n relaciones, hay $(2(n-1))!/(n-1)!$ órdenes de reunión diferentes (se deja el cálculo de esta expresión al lector en el Ejercicio 13.10). Para las reuniones que implican números pequeños de relaciones, este número resulta aceptable; por ejemplo, con $n = 5$, el número es 1.680. Sin embargo, a medida que n se incrementa, este número crece rápidamente. Con $n = 7$, el número es 665.280; con $n = 10$, ¡el número es mayor de 17.600 millones!

Afortunadamente, no es necesario generar todas las expresiones equivalentes a la expresión dada. Por ejemplo, suponga que se desea hallar el mejor orden de reunión de la forma:

$$(r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5$$

que representa todos los órdenes de reunión en que r_1, r_2 y r_3 se reúnen primero (en algún orden), y el resultado se reúne (en algún orden) con r_4 and r_5. Hay doce órdenes de reunión diferentes para calcular $r_1 \bowtie r_2 \bowtie r_3$, y otros doce órdenes para calcular la reunión de este resultado con r_4 y r_5. Por tanto, parece que hay 144 órdenes de reunión que examinar. Sin embargo, una vez hallado el mejor orden de reunión para el subconjunto de relaciones $\{r_1, r_2, r_3\}$, se puede usar ese orden para las reuniones posteriores con r_4 and r_5, y se pueden ignorar todos los órdenes de reunión más costosos de $r_1 \bowtie r_2 \bowtie r_3$. Por lo que, en lugar de examinar 144 opciones, solo hace falta examinar $12 + 12$ opciones.

Usando esta idea se puede desarrollar un algoritmo de *programación dinámica* para hallar los órdenes de reunión óptimos. Los algoritmos de programación dinámica almacenan los resultados de los cálculos y los vuelven a usar; un procedimiento que puede reducir enormemente el tiempo de ejecución.

En la Figura 13.7 aparece un procedimiento recursivo que implementa el algoritmo de programación dinámica. El procedimiento selecciona relaciones individuales lo antes posible, es decir, cuando se accede a las relaciones. Resulta más sencillo entender el procedimiento suponiendo que todas las reuniones son reuniones naturales, aunque el procedimiento funciona sin cambios con cualquier condición de reunión. Con condiciones de reunión arbitrarias, se supone que las uniones de las dos subexpresiones incluyen todas las condiciones de unión que relacionan atributos de las dos subexpresiones.

El procedimiento almacena los planes de evaluación que calcula en el array asociativo *mejorplan*, que está indexado por conjuntos de relaciones. Cada elemento del array asociativo contiene dos componentes: el coste del mejor plan de S y el propio plan. El valor de *mejorplan*[S].*coste* se supone que se inicializa como ∞ si *mejorplan*[S] no se ha calculado todavía.

El procedimiento comprueba en primer lugar si el mejor plan para calcular la reunión del conjunto de relaciones dado S se ha calculado ya (y se ha almacenado en el array asociativo *mejorplan*); si es así, devuelve el plan ya calculado.

Si S contiene solo una relación, se registra en *mejorplan* la mejor forma de acceder a S (teniendo en cuenta las selecciones sobre S, si las hay). Esto puede implicar el uso de un índice para identificar las tuplas, y después buscar las tuplas (conocido frecuentemente como *exploración del índice*), o explorar la relación completa (conocido frecuentemente como *exploración de la relación*).[1]

```
procedure HallarMejorPlan(S)
  if (mejorplan[S].coste ≠ ∞) /* mejorplan[S] ya se ha calculado*/
    return mejorplan[S]
  if (S contiene solo 1 relación)
    establecer mejorplan[S].plan y mejorplan[S].coste
      según la mejor forma de acceder a S
  else for each subconjunto no vacío S1 de S tal que S1 ≠ S
    P1 = HallarMejorPlan(S1)
    P2 = HallarMejorPlan(S − S1)
    A = mejor algoritmo para reunir los resultados de P1 y P2
    coste = P1.coste + P2.coste + coste de A
    if coste < mejorplan[S].coste
      mejorplan[S].coste = coste
      mejorplan[S].plan = «ejecutar P1.plan; ejecutar P2.plan;
        reunión de resultados de P1 y P2 usando A»
  return mejorplan[S]
```

Figura 13.7. Algoritmo de programación dinámica para la optimización del orden de la reunión.

1 Si un índice contiene todos los atributos de la relación que se usan en la consulta, es posible realizar una *exploración solo del índice* que recupera los valores de atributo requeridos del índice sin obtener las tuplas reales.

Si existe alguna condición de selección en S, distinta de la que se realiza durante una exploración del índice, se añade una operación de selección para asegurar que se satisfacen todas las selecciones sobre S.

En caso contrario, si S contiene más de una relación, el procedimiento prueba todas las maneras posibles de dividir S en dos subconjuntos disjuntos. Para cada división, el procedimiento halla de manera recursiva los mejores planes para cada uno de los dos subconjuntos y luego calcula el coste del plan global usando esa división.[2] El procedimiento escoge el plan de menor coste de entre todas las alternativas de dividir S en dos conjuntos. El procedimiento almacena el plan de menor coste y su coste en el array *mejorplan* y los devuelve. La complejidad temporal del procedimiento puede demostrarse que es $O(3^n)$ (véase el Ejercicio práctico 13.11).

En realidad, el orden en que la reunión de un conjunto de relaciones genera las tuplas también es importante para hallar el mejor orden global de reunión, ya que puede afectar al coste de las reuniones posteriores (por ejemplo, si se usa una reunión por mezcla). Se dice que un orden determinado de las tuplas es un **orden interesante** si puede resultar útil para alguna operación posterior. Por ejemplo, la generación del resultado de $r_1 \bowtie r_2 \bowtie r_3$ ordenado según los atributos comunes con r_4 o r_5 puede resultar útil, pero generarlo ordenado según los atributos comunes solamente con r_1 y r_2 no resulta útil. El uso de la reunión por mezcla para calcular $r_1 \bowtie r_2 \bowtie r_3$ puede resultar más costoso que emplear algún otro tipo de técnica de reunión, pero puede que proporcione un resultado ordenado según un orden interesante.

Por tanto, no basta con hallar el mejor orden de reunión para cada subconjunto del conjunto de n relaciones dadas. Por el contrario, hay que hallar el mejor orden de reunión para cada subconjunto para cada orden interesante de la reunión resultante para ese subconjunto. El número de subconjuntos de n relaciones es 2^n. El número de criterios de ordenación interesantes no suele ser grande. Así, hay que almacenar alrededor de 2^n expresiones de reunión. El algoritmo de programación dinámica para hallar el mejor orden de reunión puede extenderse de manera sencilla para que trabaje con los criterios de ordenación. El coste del algoritmo extendido depende del número de órdenes interesantes para cada subconjunto de relaciones; dado que se ha hallado que este número en la práctica es pequeño, el coste se queda en $O(3^n)$. Con $n = 10$, este número es de alrededor de 59.000, que es mucho mejor que los 17.600 millones de órdenes de reunión diferentes. Y lo que es más importante, el almacenamiento necesario es mucho menor que antes, ya que solo hace falta almacenar un orden de reunión por cada orden interesante de cada uno de los 1.024 subconjuntos de $r_1, \ldots, r_{10}$. Aunque los dos números siguen creciendo rápidamente con n, las reuniones que se producen con frecuencia suelen tener menos de diez relaciones y pueden manejarse con facilidad.

13.4.2. Optimización basada en coste con reglas de equivalencia

La técnica de optimización del orden de reunión que se acaba de ver anteriormente maneja la mayor parte de las consultas que realizan una reunión interna de un conjunto de relaciones. Sin embargo, muchas consultas usan otras funciones, como la agregación, la reunión externa y las consultas anidadas, que no se tratan en la selección de orden de la reunión.

Muchos optimizadores utilizan sus propias heurísticas de transformación para construir otras reuniones y aplican el algoritmo de selección del orden de la reunión basado en coste a las subexpresiones que tienen solo reunión y selección. Los detalles sobre estas heurísticas son en su mayor parte específicos de los distintos optimizadores y no se van a tratar. Sin embargo, las transformaciones heurísticas para las consultas anidadas se utilizan ampliamente y se consideran con más detalle en la Sección 13.4.4.

En esta sección, sin embargo, se verá cómo crear un optimizador de coste de propósito general basado en reglas de equivalencia, que pueden manejar una amplia variedad de constructores de consultas.

La ventaja de usar reglas de equivalencia es que resulta fácil extender el optimizador con nuevas reglas para manejar distintas construcciones de consultas. Por ejemplo, las consultas anidadas se pueden representar mediante las construcciones del álgebra relacional extendida y las transformaciones de las consultas anidadas se pueden expresar como reglas de equivalencia. Ya se han visto las reglas de equivalencia con operaciones de agregación y también se pueden crear para la reunión externa.

En la Sección 13.2.4 ya se vio que un optimizador puede generar sistemáticamente todas las expresiones equivalentes a una consulta dada.

El procedimiento de generación de expresiones equivalentes se puede modificar para que genere todos los posibles planes de evaluación de la siguiente forma: se añaden una nueva clase de reglas de equivalencia, llamadas **reglas de equivalencia física,** que permiten transformar una operación física como una reunión asociativa o una reunión de bucle anidado en una operación lógica. Con estas reglas, junto al conjunto original de reglas de equivalencia, el procedimiento puede generar todos los planes de evaluación posibles. Se pueden usar las técnicas de evaluación de coste vistas anteriormente para elegir el plan óptimo (el de menor coste).

Sin embargo, el procedimiento que se muestra en la Sección 13.2.4 es muy costoso, incluso si no se considera la generación de planes de evaluación. Para que esta forma resulte más eficiente se necesita:

1. Una representación eficiente en espacio que evite realizar múltiples copias de la misma subexpresión, cuando se apliquen las reglas.
2. Una técnica eficiente para detectar derivaciones duplicadas de la misma subexpresión.
3. Una forma de programación dinámica, basada en la **memoización,** que almacene el plan de evaluación de consulta óptima para una subexpresión cuando se optimiza por primera vez; las siguientes peticiones para optimizar la misma subexpresión se manejan retornando al plan ya memoizado.
4. Técnicas que eviten la generación de todos los posibles planes, manteniendo un registro de los de menor coste generados para las subexpresiones hasta el momento, y eliminando cualquier plan que sea de mayor coste que el de menor coste encontrado hasta el momento.

Los detalles son de mayor complejidad que los que se desean tratar. Esta aproximación fue pionera en el proyecto de investigación Volcano, y el optimizador de consultas de SQL Server se basa en esta aproximación. Consulte las notas bibliográficas para más referencias.

2 Fíjese que una reunión de bucle anidado indexada se considera para la reunión de $P1$ y $P2$, con $P2$ como relación interna, si $P2$ solo tiene una relación, digamos r, y existe un índice sobre los atributos de r. El plan $P2$ puede contener un acceso indexado a r, según las condiciones de selección de r. Para permitir el uso de la reunión de bucle anidado indexado usando la condición de selección sobre r se debería eliminar de $P2$; si no, se debería comprobar la condición de selección en las tuplas que devuelva el índice de los atributos de reunión de r.

13.4.3. Heurísticas en la optimización

Un inconveniente de la optimización basada en el coste es el coste de la propia optimización. Aunque el coste de la optimización de las consultas puede reducirse mediante algoritmos inteligentes, el número de planes de evaluación distintos para una consulta puede ser muy grande y buscar el plan óptimo a partir de este conjunto requiere un gran esfuerzo de cómputo. Por ello, muchos sistemas usan **heurísticas** para reducir el coste de la optimización.

Un ejemplo de regla heurística es la siguiente regla para la transformación de consultas del álgebra relacional:

- Realizar las operaciones de selección tan pronto como sea posible.

Los optimizadores heurísticos usan esta regla sin averiguar si se reduce el coste mediante esta transformación. En el primer ejemplo de transformación de la Sección 13.2 se forzó la operación selección en una reunión.

Se dice que la regla anterior es heurística porque suele ayudar a reducir el coste, aunque no lo haga siempre. Como ejemplo de dónde puede dar lugar a un incremento del coste, considérese una expresión $\sigma_\theta\ (r \bowtie s)$, donde la condición θ solo hace referencia a atributos de s. Ciertamente, la selección puede realizarse antes que la reunión. Sin embargo, si r es extremadamente pequeña comparada con s, y si hay un índice basado en los atributos de reunión de s pero no hay ningún índice basado en los atributos usados por θ, probablemente resulte una mala idea llevar a cabo la selección pronto. Realizar pronto la selección (es decir, directamente sobre s) exigiría hacer una exploración de todas las tuplas de s. Es posible que resulte menos costoso calcular la reunión usando el índice y luego rechazar las tuplas que no superen la selección.

La operación proyección, como la operación selección, reduce el tamaño de las relaciones. Por tanto, siempre que haya que generar una relación temporal, resulta ventajoso aplicar inmediatamente cuantas proyecciones sea posible. Esta ventaja sugiere un acompañante a la heurística de «realizar las selecciones tan pronto como sea posible»:

- Realizar las proyecciones pronto.

Suele resultar mejor llevar a cabo las selecciones antes que las proyecciones, ya que las selecciones tienen la posibilidad de reducir mucho el tamaño de las relaciones y permiten el empleo de índices para tener acceso a las tuplas. Un ejemplo parecido al utilizado para la heurística de selección debería convencer al lector de que esta heurística no siempre reduce el coste.

La mayoría de los optimizadores de consultas tienen más heurísticas para reducir el coste de la optimización. Por ejemplo, muchos optimizadores de consultas, como el optimizador de System R,[3] no toman en consideración todos los órdenes de reunión, sino que restringen la búsqueda a tipos concretos de órdenes de reunión. El optimizador de System R solo toma en consideración los órdenes de reunión en los que el operando de la derecha de cada reunión es una de las relaciones iniciales $r_1, \ldots, r_n$. Estos órdenes de reunión se denominan **órdenes de reunión en profundidad por la izquierda.** Los órdenes de reunión en profundidad por la izquierda resultan especialmente convenientes para la evaluación encauzada, ya que el operando de la derecha es una relación almacenada y, así, solo se encauza una entrada por cada reunión.

La Figura 13.8 muestra la diferencia entre un árbol de reunión en profundidad por la izquierda y otro que no lo es. El tiempo que se tarda en tomar en considerar todos los órdenes de reunión en profundidad por la izquierda es $O(n!)$, que es mucho menor que el tiempo necesario para tomar en consideración todos los órdenes de reunión. Con el empleo de las optimizaciones de programación dinámica, el optimizador de System R puede encontrar el mejor orden de reunión en un tiempo de $O(n2^n)$. Compare este coste con el tiempo de $O(3^n)$ necesario para encontrar el mejor orden de reunión global. El optimizador de System R usa heurísticas para lanzar las selecciones y proyecciones hacia abajo en el árbol de consultas.

El enfoque heurístico para reducir el coste de la selección del orden de reunión, que se usó originalmente en algunas versiones de Oracle, funciona básicamente de esta manera: para cada reunión de grado n toma en consideración n planes de evaluación. Cada plan usa un orden de reunión en profundidad por la izquierda, comenzando con una relación diferente de las n existentes. La heurística crea el orden de reunión para cada uno de los n planes de evaluación seleccionando de manera repetida la «mejor» relación que reunir a continuación, con base en la clasificación de los caminos de acceso disponibles. Se escoge la reunión en bucle anidado o mezcla-ordenación para cada una de las reuniones, en función de los caminos de acceso disponibles. Finalmente, la heurística escoge uno de los n planes de evaluación de manera heurística, basada en la minimización del número de reuniones de bucle anidado que no tienen disponible un índice para la relación interna y en el número de reuniones por mezcla-ordenación.

Los enfoques de la optimización de consultas que integran la selección heurística y la generación de planes de acceso alternativos se han adoptado en varios sistemas. El enfoque usado en System R y en su sucesor, el proyecto Starburst, es un procedimiento jerárquico basado en el concepto de bloques anidados de SQL. Las técnicas de optimización basadas en costes aquí descritas se usan por separado para cada bloque de la consulta. Los optimizadores de varios productos de bases de datos, como DB2 de IBM y Oracle, se basan en este enfoque, con extensiones para tratar otras operaciones como la agregación. Para las consultas compuestas de SQL (que usan la operación $\cup$, $\cap$ o $-$), el optimizador procesa cada componente por separado y combina los planes de evaluación para formar el plan global de evaluación.

La mayoría de los optimizadores permiten especificar un presupuesto de coste para la optimización de consultas. La búsqueda del plan óptimo termina cuando se supera este **presupuesto de coste de optimización** devolviendo el mejor plan encontrado hasta el momento. El propio presupuesto se puede establecer dinámicamente; por ejemplo, si se encuentra un plan de menor coste para una consulta, se puede reducir el presupuesto bajo la premisa de que no se use demasiado tiempo en optimizar una consulta si el mejor plan que se encuentre ya es suficientemente de bajo coste. Por otra parte, si el mejor plan que se ha encontrado tiene un alto coste, tiene sentido invertir más tiempo en la optimización, lo que puede conducir a una reducción significativa del tiempo de ejecución. Para aprovechar mejor esta idea, los optimizadores normalmente aplican primero heurísticas sencillas para encontrar un plan y, posteriormente, siguen con las optimizaciones completas basadas en coste con un presupuesto basado en el plan heurísticamente elegido.

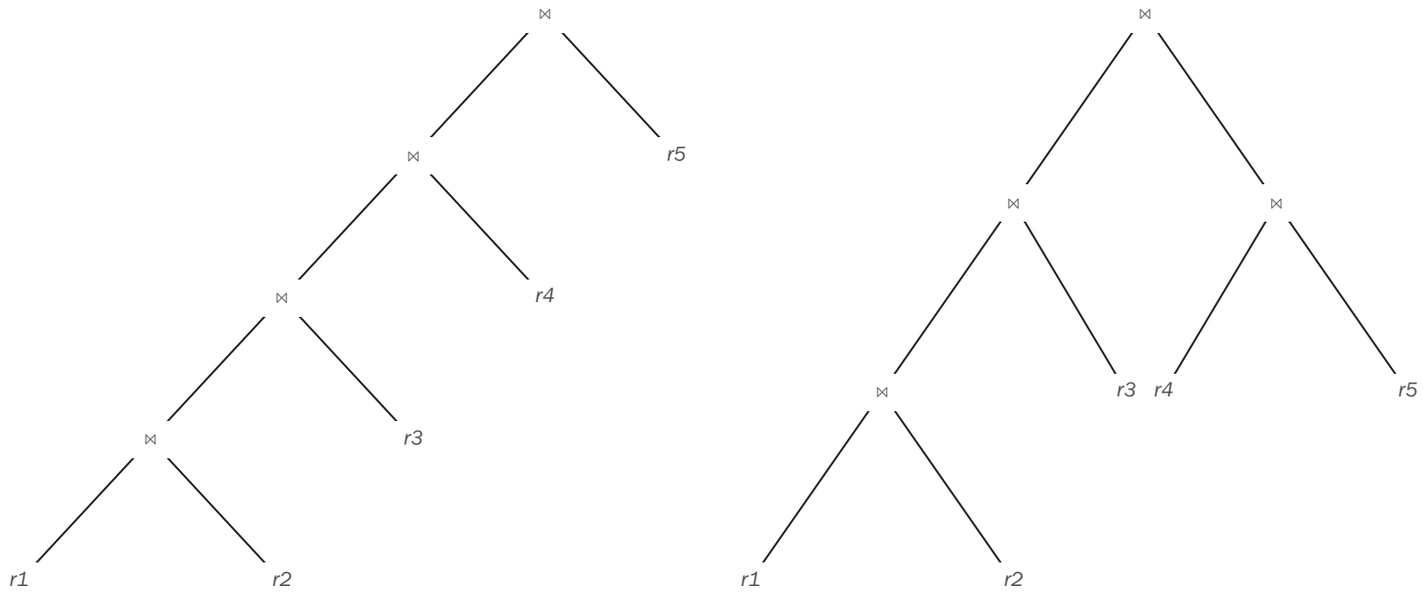
Muchas aplicaciones ejecutan la misma consulta repetidamente pero con diferentes valores de las constantes. Por ejemplo, una aplicación de una universidad puede ejecutar repetidamente una consulta para buscar las asignaturas en las que se ha matriculado un estudiante, pero para distintos estudiantes que tienen diversos valores de *ID*.[4] Como heurística, muchos optimizadores optimizan una vez la consulta con los valores de la primera ejecución, y almacenan en la caché el plan de la consulta. Cada vez que se ejecuta de nuevo la consulta, quizás con nuevos valores de las constantes, se

3 System R fue la primera implementación de SQL, y su optimizador fue pionero en la idea de optimización de orden de reunión basada en coste.

4 Para la consulta de matrícula de un estudiante, el plan debería ser realmente el mismo para cualquier ID de estudiante. Pero una consulta que tomara un intervalo de ID de estudiantes y devolviera la información de matrícula de todos los ID de estudiantes en el intervalo, probablemente tendría un plan óptimo diferente si el intervalo fuera pequeño que si el intervalo fuera grande.

reutiliza el plan de la consulta en la caché (obviamente usando los nuevos valores de las constantes). El plan óptimo para las nuevas constantes puede diferir del plan óptimo para los valores iniciales, pero como heurística se reutiliza el plan guardado en la caché. La reutilización y uso de la caché para los planes de consultas se denomina **caché de planes.**

Incluso con el uso de la heurística, la optimización de consultas basada en los costes impone una sobrecarga sustancial al procesamiento de las consultas. No obstante, el coste añadido de la optimización de las consultas basada en los costes suele compensarse con creces por el ahorro en tiempo de ejecución de la consulta, que queda dominado por el lento acceso a los discos. La diferencia en el tiempo de ejecución entre un buen plan y uno malo puede ser enorme, lo cual hace que la optimización de las consultas sea esencial. El ahorro conseguido se multiplica en las aplicaciones que se ejecutan de manera regular, en las que se puede optimizar la consulta una sola vez y usar el plan de consultas seleccionado cada vez que se ejecute la consulta. Por tanto, la mayor parte de los sistemas comerciales incluyen optimizadores relativamente sofisticados. En las notas bibliográficas puede obtener referencias de las descripciones de los optimizadores de consultas de los sistemas de bases de datos reales.



(a) Árbol de reunión en profundidad por la izquierda

(b) Árbol de reunión que no es en profundidad por la izquierda

Figura 13.8. Árboles de reunión en profundidad por la izquierda.

13.4.4. Optimización de subconsultas anidadas**

SQL trata conceptualmente a las subconsultas anidadas de la cláusula **where** como funciones que toman parámetros y devuelven un solo valor o un conjunto de valores (quizás, un conjunto vacío). Los parámetros son las variables de la consulta del nivel externo que se usan en la subconsulta anidada (estas variables se denominan **variables de correlación**). Por ejemplo, suponga que se tiene la siguiente consulta:

select *nombre*
from *profesor*
where exists (**select** *
  from *enseña*
  where *profesor.ID* = *enseña.ID*
    and *enseña.año* = 2007);

Conceptualmente, la subconsulta puede considerarse como una función que toma un parámetro (aquí, *profesor.ID*) y devuelve el conjunto de todas las asignaturas que se enseñaron en el 2007 por profesores (con el mismo *ID*).

SQL evalúa la consulta global (conceptualmente) calculando el producto cartesiano de las relaciones de la cláusula **from** externa y comprobando luego los predicados de la cláusula **where** para cada tupla del producto. En el ejemplo anterior, el predicado comprueba si el resultado de la evaluación de la subconsulta está vacío.

Esta técnica para evaluar una consulta con una subconsulta anidada se denomina **evaluación correlacionada.** La evaluación correlacionada no resulta muy eficiente, ya que la subconsulta se evalúa por separado para cada tupla de la consulta del nivel externo. Puede dar lugar a gran número de operaciones aleatorias de E/S a disco.

Por tanto, los optimizadores de SQL intentan transformar las subconsultas anidadas en reuniones, siempre que resulte posible. Los algoritmos de reunión eficientes evitan las costosas operaciones aleatorias de E/S. Cuando no es posible realizar la transformación, el optimizador conserva las subconsultas como expresiones independientes, las optimiza por separado y luego las evalúa mediante la evaluación correlacionada.

Como ejemplo de transformación de una subconsulta anidada en una reunión, la consulta del ejemplo anterior puede reescribirse como:

select *nombre*
from *profesor*, *enseña*
where *profesor.ID* = *enseña.ID* **and** *enseña.año* = 2007;

(Para reflejar correctamente la semántica de SQL no debe cambiar el número de derivaciones duplicadas debido a la reescritura; la consulta reescrita puede modificarse para asegurarse de que se cumple esta propiedad, como se verá en breve.)

En el ejemplo, la subconsulta anidada era muy sencilla. En general, puede que no resulte posible desplazar las relaciones de la subconsulta anidada a la cláusula **from** de la consulta externa. En su lugar, se crea una relación temporal que contiene el resultado de la consulta anidada *sin* las selecciones empleando las variables de correlación de la consulta externa y se reúne la tabla temporal con la consulta del nivel exterior. Por ejemplo, una consulta de la forma:

select …

from L_1

where P_1 **and exists** (**select** *
from L_2
where P_2);

donde P_2 es una conjunción de predicados más sencillos, puede reescribirse como:

create table t_1 **as**

select distinct V
from L_2
where P_2^1;

select …

from L_1, t_1

where P_1 **and** $P\ P_2^2$;

donde P_2^1 contiene los predicados de P_2 sin las selecciones que implican las variables de correlación, y P_2^2 reintroduce las selecciones que implican las variables de correlación (con las relaciones a que se hace referencia en el predicado renombradas adecuadamente). Aquí V contiene todos los atributos que se usan en las selecciones con las variables de correlación en la subconsulta anidada.

En el ejemplo, la consulta original se habría transformado en:

create table t_1 **as**
select distinct *ID*
from *enseña*
where *año* $= 2007$;

select *nombre*

from *profesor*, t_1

where t_1 *.ID* $=$ *profesor.ID*;

La consulta que se reescribió para mostrar la creación de relaciones temporales se puede obtener simplificando la consulta transformada anteriormente, suponiendo que no importe el número de duplicados de cada tupla.

El proceso de sustituir una consulta anidada por una consulta con una reunión (acaso con una relación temporal) se denomina **descorrelación.**

La descorrelación resulta más complicada cuando la subconsulta anidada usa agregación o cuando el resultado de la subconsulta anidada se usa para comprobar la igualdad, o cuando la condición que enlaza la subconsulta anidada con la consulta exterior es **not exists,** etc. No se intentará dar algoritmos para el caso general y, en su lugar, se remitirá al lector a los elementos de importancia en las notas bibliográficas.

La optimización de las subconsultas anidadas complejas es una labor difícil, como puede deducirse del estudio anterior, y muchos optimizadores solo llevan a cabo una cantidad limitada de descorrelación. Resulta más conveniente evitar el empleo de subconsultas anidadas complejas, siempre que sea posible, ya que no se puede asegurar que el optimizador de consultas tenga éxito en su conversión a una forma que pueda evaluarse de manera eficiente.

13.5. Vistas materializadas**

Cuando se define una vista, normalmente la base de datos solo almacena la consulta que define la vista. Por el contrario, una **vista materializada** es una vista cuyo contenido se calcula y se almacena. Las vistas materializadas constituyen datos redundantes, en el sentido de que su contenido puede deducirse de la definición de la vista y del resto del contenido de la base de datos. No obstante, en muchos casos resulta mucho menos costoso leer el contenido de una vista materializada que calcular el contenido de la vista ejecutando la consulta que la define.

Las vistas materializadas resultan importantes para la mejora del rendimiento de algunas aplicaciones. Considere esta vista, que devuelve el sueldo total de cada departamento:

create view *departamento_total_sueldo* (*nombre_dept*, *total_sueldo*) **as**

select *nombre_ dept*, **sum** (*sueldo*)

from *profesor*

group by *nombre_dept*;

Suponga que se solicita con frecuencia el sueldo total de los departamentos. El cálculo de la vista exige la lectura de cada tupla de *profesor* correspondiente a un departamento y la suma de todas las cuantías salariales, lo que puede llevar mucho tiempo. En cambio, si la definición de la vista del importe total de los sueldos estuviera materializada, las cuantías salariales totales se podrían encontrar buscando una sola tupla de la vista materializada.[5]

13.5.1. Mantenimiento de las vistas

Un problema con las vistas materializadas es que hay que mantenerlas actualizadas cuando se modifican los datos empleados en la definición de la vista. Por ejemplo, si se actualiza el valor del *sueldo* de un profesor, la vista materializada se vuelve inconsistente con los datos subyacentes y hay que actualizarla. La tarea de mantener actualizada una vista materializada con los datos subyacentes se denomina **mantenimiento de la vista.**

Las vistas se pueden mantener mediante código escrito a mano: es decir, cada fragmento del código que actualiza el valor del *sueldo* se puede modificar para que actualice también el sueldo total del departamento correspondiente. Sin embargo, este enfoque es propenso a errores, ya que es fácil olvidarse de hacerlo en algún momento cuando se actualice el *sueldo* y la vista materializa ya no volverá a ser consistente con los datos subyacentes.

Otra opción para el mantenimiento de las vistas materializadas es la definición de disparadores para la inserción, el borrado y la actualización de cada relación de la definición de la vista. Los disparadores deben modificar el contenido de la vista materializada para tener en cuenta el cambio que ha provocado que se active el disparador. Una manera sencilla de hacerlo es volver a calcular completamente la vista materializada con cada actualización.

Una opción mejor consiste en modificar solo las partes afectadas de la vista materializada, lo que se conoce como **mantenimiento incremental de la vista.** En la Sección 13.5.2 se describe la manera de llevar a cabo el mantenimiento incremental de la vista.

Los sistemas modernos de bases de datos proporcionan más soporte directo para el mantenimiento incremental de las vistas. Los programadores de bases de datos ya no necesitan definir dispara-

5 La diferencia puede que no sea muy grande para una universidad de tamaño medio, pero en otras situaciones la diferencia puede ser muy importante. Por ejemplo, si la vista materializada calcula las ventas totales por producto, de una relación de ventas con decenas de millones de tuplas, la diferencia entre calcular el agregado de los datos subyacentes y buscar en la vista materializada puede ser de varios órdenes de magnitud.

dores para el mantenimiento de las vistas. Por el contrario, cuando una vista se declara materializada, el sistema de bases de datos calcula su contenido y actualiza este de manera incremental cuando se modifican los datos subyacentes.

La mayoría de los sistemas de bases de datos realizan un **mantenimiento inmediato de las vistas;** es decir, se realiza el mantenimiento incremental tan pronto como se produce una actualización como parte de una transacción de actualización. Algunos sistemas de bases de datos también permiten el **mantenimiento diferido de las vistas,** en el que se difiere el mantenimiento a un momento posterior; por ejemplo, las actualizaciones se pueden recopilar a lo largo del día y las vistas materializadas se pueden actualizar durante la noche. Este enfoque reduce la sobrecarga en las transacciones de actualización. Sin embargo, es posible que las vistas materializadas con mantenimiento diferido no sean consistentes con las relaciones sobre las que están definidas.

13.5.2. Mantenimiento incremental de las vistas

Para comprender el modo de mantener de manera incremental las vistas materializadas se comenzará por considerar las operaciones individuales y luego se verá la manera de manejar una expresión completa.

Los cambios en una relación que puedan hacer que se quede desactualizada una vista materializada son las inserciones, los borrados y las actualizaciones. Para simplificar la descripción se sustituyen las actualizaciones por el borrado de la tupla seguida de la inserción de la tupla actualizada. Por tanto, solo hay que considerar las inserciones y los borrados. Los cambios (inserciones y borrados) en la relación o en la expresión se conocen como su **diferencial.**

13.5.2.1. La operación reunión

Considere la vista materializada $v = r \bowtie s$. Suponga que se modifica r insertando un conjunto de tuplas denotado por i_r. Si el valor antiguo de r se denota por r^{vieja} y el valor nuevo de r por r^{nueva}, $r^{nueva} = r^{vieja} \cup i_r$. Ahora bien, el valor antiguo de la vista, v^{vieja} viene dado por $r^{vieja} \bowtie s$, y el valor nuevo v^{nueva} viene dado por $r^{nueva} \bowtie s$. Se puede reescribir $r^{nueva} \bowtie s$ como $(r^{vieja} \cup i_r) \bowtie s$, lo que se puede reescribir una vez más como $(r^{vieja} \bowtie s) \cup (i_r \bowtie s)$. En otras palabras:

$$v^{nueva} = v^{vieja} \cup (i_r \bowtie s)$$

Por tanto, para actualizar la vista materializada v, solo hace falta añadir las tuplas $i_r \bowtie s$ al contenido antiguo de la vista materializada. Las inserciones en s se manejan de una manera completamente simétrica.

Suponga ahora que se modifica r borrando un conjunto de tuplas denotado por d_r. Usando el mismo razonamiento que anteriormente, se obtiene:

$$\mathrm{v}^{nueva} = \mathrm{v}^{vieja} - (\mathrm{d}_r \bowtie \mathrm{s})$$

Las operaciones de borrado en s se manejan de una manera completamente simétrica.

13.5.2.2. Las operaciones selección y proyección

Considérese una vista $v = \sigma_\theta (r)$. Si se modifica r insertando un conjunto de tuplas i_r, el nuevo valor de v puede calcularse como:

$$v^{nueva} = v^{vieja} \cup \sigma_\theta (i_r)$$

De manera parecida, si se modifica r borrando un conjunto de tuplas d_r, el valor nuevo de v puede calcularse como:

$$v^{nueva} = v^{vieja} - \sigma_\theta (d_r)$$

La proyección es una operación más difícil de tratar. Considere una vista materializada $v = \Pi_A(r)$. Suponga que la relación r está en el esquema $R = (A, B)$ y que r contiene dos tuplas, $(a, 2)$ y $(a, 3)$. Entonces, $\Pi_A(r)$ tiene una sola tupla, (a). Si se borra la tupla $(a, 2)$ de r, no se puede borrar la tupla (a) de $\Pi_A(r)$: si se hiciera, el resultado sería una relación vacía, mientras que en realidad $\Pi_A(r)$ sigue teniendo una tupla (a). El motivo es que la misma tupla (a) se obtiene de dos maneras, y que el borrado de una tupla de r solo borra una de las formas de obtener (a); la otra sigue presente.

Este motivo también ofrece una pista de la solución: para cada tupla de una proyección como $\Pi_A(r)$, se lleva la cuenta del número de veces que se ha obtenido.

Cuando se borra un conjunto de tuplas d_r de r, para cada tupla t de d_r hay que hacer lo siguiente. Sea $t.A$ la proyección de t sobre el atributo A. Se busca $(t.A)$ en la vista materializada y se disminuye la cuenta almacenada con ella en 1. Si la cuenta llega a 0, se borra $(t.A)$ de la vista materializada.

El manejo de las inserciones resulta relativamente directo. Cuando un conjunto de tuplas i_r se inserta en r, para cada tupla t de i_r se hace lo siguiente: si $(t.A)$ ya está presente en la vista materializada, se incrementa la cuenta almacenada con ella en 1. En caso contrario, se añade $(t.A)$ a la vista materializada con la cuenta puesta a 1.

13.5.2.3. Las operaciones de agregación

Las operaciones de agregación se comportan aproximadamente como las proyecciones. Las operaciones de agregación en SQL son **count, sum, avg, min** y **max:**

- **count:** considere la vista materializada $v = {}_A\mathcal{G}_{count(B)}(r)$, que calcula la cuenta del atributo B, después de agrupar r según el atributo A.

  Cuando se inserta un conjunto de tuplas i_r en r, para cada tupla t de i_r se realiza lo siguiente: se busca el grupo $t.A$ en la vista materializada. Si no se halla presente, se añade $(t.A, 1)$ a la vista materializada. Si el grupo $t.A$ se halla presente, se añade 1 a su cuenta.

  Cuando un conjunto d_r se borra de r, para cada tupla t de d_r se realiza lo siguiente: se busca el grupo $t.A$ en la vista materializada y se resta 1 de la cuenta del grupo. Si la cuenta llega a 0, se borra la tupla para el grupo $t.A$ de la vista materializada.

- **sum:** considere la vista materializada $v = {}_A\mathcal{G}_{sum(B)}(r)$.

  Cuando un conjunto de tuplas i_r se inserta en r, para cada tupla t de i_r se realiza lo siguiente: se busca el grupo $t.A$ en la vista materializada. Si no se halla presente, se añade $(t.A, t.B)$ a la vista materializada; además, se almacena una cuenta de 1 asociada con $(t.A, t.B)$, igual que se hizo para las proyecciones. Si el grupo $t.A$ se halla presente, se añade el valor de $t.B$ al valor agregado para el grupo y se añade 1 a su cuenta.

  Cuando se borra un conjunto de tuplas d_r de r, para cada tupla t de d_r se realiza lo siguiente: se busca el grupo $t.A$ en la vista materializada y se resta $t.B$ del valor agregado para el grupo. También se resta 1 de la cuenta del grupo y, si la cuenta llega a 0, se borra la tupla para el grupo $t.A$ de la vista materializada.

  Si no se guardara el valor de cuenta adicional no se podría distinguir el caso de que la suma para el grupo sea 0 del caso en que se ha borrado la última tupla de un grupo.

- **avg:** considere la vista materializada $v = {}_A\mathcal{G}_{avg(B)}(r)$.

  La actualización directa del promedio de una inserción o de un borrado no resulta posible, ya que no solo depende del promedio antiguo y de la tupla que se inserta o borra, sino también del número de tuplas del grupo.

  En lugar de eso, para tratar el caso de **avg** se conservan los valores de agregación **sum** y **count** como se describieron anteriormente y se calcula el promedio como la suma dividida por la cuenta.

- **min, max:** considere la vista materializada $v = {}_A\mathcal{G}_{min(B)}(r)$ (el caso de **max** es completamente equivalente).

  El tratamiento de las inserciones en r es inmediato. La conservación de los valores de agregación **min** y **max** para los borrados puede resultar más costosa. Por ejemplo, si se borra de r la tupla correspondiente al valor mínimo para un grupo, hay que examinar las demás tuplas de r que están en el mismo grupo para hallar el nuevo valor mínimo.

13.5.2.4. Otras operaciones

La operación de conjuntos *intersección* se mantiene de la siguiente forma: dada la vista materializada $v = r \cap s$, cuando una tupla se inserta en r se comprueba si está presente en s y, en caso afirmativo, se añade a v. Si se borra una tupla de r, se borra de la intersección si se halla presente. Las otras operaciones con conjuntos, *unión* y *diferencia de conjuntos*, se tratan de manera parecida; los detalles se dejan al lector.

Las reuniones externas se tratan de manera muy parecida a las reuniones, pero con algún trabajo adicional. En el caso del borrado de r, hay que manejar las tuplas de s que ya no coinciden con ninguna tupla de r. En el caso de una inserción en r, hay que manejar las tuplas de s que no coincidían con ninguna tupla de r. De nuevo se dejan los detalles al lector.

13.5.2.5. Tratamiento de expresiones

Hasta ahora se ha visto el modo de actualizar de manera incremental el resultado de una sola operación. Para tratar una expresión entera se pueden obtener expresiones para el cálculo del cambio incremental en el resultado de cada subexpresión, comenzando por las de menor tamaño.

Por ejemplo, suponga que se desea actualizar de manera incremental una vista materializada $E_1 \bowtie E_2$ cuando se inserta un conjunto de tuplas i_r en la relación r. Suponga que r se usa solo en E_1. Suponga que el conjunto de tuplas que se va a insertar en E_1 viene dado por la expresión D_1. Entonces, la expresión $D_1 \bowtie E_2$ da el conjunto de tuplas que hay que insertar en $E_1 \bowtie E_2$.

Consulte las notas bibliográficas para obtener más detalles sobre la conservación incremental de las vistas con expresiones.

13.5.3. Optimización de consultas y vistas materializadas

La optimización de consultas puede llevarse a cabo tratando a las vistas materializadas igual que a las relaciones normales. No obstante, las vistas materializadas ofrecen más oportunidades para la optimización:

- Reescritura de las consultas para el empleo de vistas materializadas:

  Suponga que está disponible la vista materializada $v = r \bowtie s$ y que un usuario ejecuta la consulta $r \bowtie s \bowtie t$. Puede que la reescritura de la consulta como $v \bowtie t$ proporcione un plan de consulta más eficiente que la optimización de la consulta tal y como se ha enviado. Por tanto, es función del optimizador de consultas reconocer si se puede usar una vista materializada para acelerar una consulta.
- Sustitución del uso de una vista materializada por la definición de la vista:

  Suponga que está disponible la vista materializada $v = r \bowtie s$, pero sin ningún índice definido sobre ella, y que un usuario envía la consulta $\sigma_{A=10}(v)$. Suponga también que s tiene un índice sobre el atributo común B, y que r tiene un índice sobre el atributo A. Puede que el mejor plan para esta consulta sea sustituir v por $r \bowtie s$, lo que puede llevar al plan de consulta $\sigma_{A=10}(r) \bowtie s$; la selección y la reunión pueden llevarse a cabo de manera eficiente empleando los índices sobre $r.A$ y sobre $s.B$, respectivamente. Por el contrario, puede que la evaluación de la selección directamente sobre v necesite una exploración completa de v, lo que puede resultar más costoso.

Las notas bibliográficas ofrecen indicaciones para investigar el modo de llevar a cabo de manera eficiente la optimización de las consultas con vistas materializadas.

13.5.4. Selección de vistas materializadas y de índices

Otro problema de optimización relacionado es el de la **selección de las vistas materializadas;** es decir, «la identificación del mejor conjunto de vistas para su materialización». Esta decisión debe tomarse con base en la **carga de trabajo** del sistema, que es una secuencia de consultas y de actualizaciones que refleja la carga típica del sistema. Un criterio sencillo sería la selección de un conjunto de vistas materializadas que minimice el tiempo global de ejecución de la carga de trabajo de consultas y de actualizaciones, incluido el tiempo empleado para conservar las vistas materializadas. Los administradores de bases de datos suelen modificar este criterio para tener en cuenta la importancia de las diferentes consultas y actualizaciones: puede ser necesaria una respuesta rápida para algunas consultas y actualizaciones, mientras que puede resultar aceptable una respuesta lenta para otras.

Los índices son como las vistas materializadas, en el sentido de que también son datos derivados, pueden acelerar las consultas y pueden ralentizar las actualizaciones. Por tanto, el problema de la **selección de índices** se halla íntimamente relacionado con el de la selección de las vistas materializadas, aunque resulta más sencillo. Se examina con más detalle la selección de índices y de vistas materializadas en las Secciones 24.1.6 y 24.1.7.

La mayoría de los sistemas de bases de datos proporcionan herramientas para ayudar a los administradores de bases de datos en la selección de los índices y de las vistas materializadas. Estas herramientas examinan el historial de consultas y de actualizaciones y sugieren los índices y las vistas que hay que materializar. Database Tuning Assistant, de SQL Server de Microsoft; Design Advisor, de DB2 de IBM y Tuning Wizard de SQL, de Oracle, son ejemplos de estas herramientas.

13.6. Temas avanzados de optimización de consultas**

Existen distintas formas de optimizar consultas, además de las que ya se han tratado. Se examinan algunas de ellas en esta sección.

13.6.1. Optimización primeros-*K*

Muchos de los resultados que se obtienen de las consultas están ordenados por algún atributo y solo requieren los primeros-*K* resultados, para cierto *K*. A veces, el límite *K* se indica explícitamente. Por ejemplo, algunas bases de datos disponen de una cláusula **limit** *K* que solo genera los primeros-*K* resultados que devuelve la consulta. Otras bases de datos disponen de formas alternativas para especificar límites similares. En otros casos, no se puede indicar un límite en la consulta, pero el optimizador permite que se le den indicaciones, que lo especifiquen, sugiriendo así que probablemente solo se necesiten obtener los primeros-*K* valores de la consulta, incluso aunque la consulta genere más resultados.

Cuando *K* es pequeño, un plan de optimización de consultas que genera un conjunto entero de resultados, los ordena y genera los primeros-*K*, es muy ineficiente, ya que descarta la mayor parte de los resultados intermedios que calcula. Se han propuesto distintas técnicas para optimizar estas *consultas primeros-K*. Un enfoque es utilizar planes encauzados que puedan generar los resultados con un orden. Otro enfoque consiste en estimar cuál será el mayor valor de los atributos ordenados que aparecerá en la salida primeros-*K*, e introducir los predicados de selección que eliminen valores mayores. Si se generan tuplas posteriores a los primeros-*K*, se descartan, y si se generan muy pocas tuplas, se cambia la condición de selección y se vuele a ejecutar la consulta. Consulte las notas bibliográficas para más referencias sobre optimización de primeros-*K* (*top-K*).

13.6.2. Minimización de reunión

Cuando las consultas se generan mediante vistas, a veces se realizan más reuniones de las necesarias para calcular la consulta. Por ejemplo, una vista *v* puede incluir la reunión de *profesor* y *departamento*, pero el uso de la relación *v* puede que solo use los atributos de *profesor*. El atributo de la unión *nombre_dept* de *profesor* es una clave externa que hace referencia a *departamento*. Suponiendo que el nombre del *profesor.dept_nombre* se ha declarado **not null,** la reunión con *departamento* se puede eliminar, sin ningún impacto en la consulta. La reunión con *departamento* no elimina ninguna tupla de *profesor*, con la suposición anterior, ni genera copias extra de ninguna tupla de *profesor*.

Eliminar una relación de una reunión, como se ha visto antes, es un ejemplo de minimización de reuniones. De hecho, esta minimización se puede realizar también en otras situaciones. Consulte las notas bibliográficas para más información sobre la minimización de reuniones.

13.6.3. Optimización de actualizaciones

Las consultas de actualización suelen requerir de subconsultas en las cláusulas **set** y **where,** que también hay que tener en cuenta cuando se actualiza la consulta. Las actualizaciones que conllevan una selección en la columna que se actualiza (por ejemplo, aumentar un 10 por ciento el salario de todos los empleados cuyo sueldo sea $\geq$ 100.000 €) hay que manejarlas con sumo cuidado. Si se realiza la actualización mientras se evalúa la selección mediante una exploración de índices, puede que se reinserte una tupla actualizada en el índice por delante de la exploración y se vuelva a revisar por el explorador; la misma tupla de empleado puede actualizarse, de forma incorrecta, más de una vez (en este caso un número infinito de veces). Un problema similar ocurre con las actualizaciones cuando tienen subconsultas cuyo resultado se ve afectado por la actualización.

El problema de una actualización que afecta a la ejecución de una consulta asociada con la actualización se conoce como **problema de Halloween** (llamado así porque fue descubierto por primera vez un día de Halloween, en IBM). El problema se puede evitar ejecutando primero las consultas que definen la actualización, creando una lista de las tuplas afectadas y actualizando las tuplas y los índices en el último paso. Sin embargo, dividir el plan de ejecución de esta forma aumenta el coste de la ejecución. Los planes de actualización se pueden optimizar comprobando si se puede producir el problema de Halloween y, si no se puede producir, realizar las actualizaciones mientras se procesa la consulta, reduciendo la sobrecarga de las actualizaciones. Por ejemplo, el problema de Halloween no puede producirse si la actualización no afecta a atributos del índice. Incluso en este caso, se puede actualizar el índice aunque la consulta se esté ejecutando, reduciendo el coste total.

Las consultas de actualización que generan un gran número de actualizaciones también se pueden optimizar recolectando las actualizaciones como una secuencia de comandos y aplicando después dichas secuencias de actualizaciones de forma separada para cada índice afectado. Cuando se aplica la secuencia de comandos de actualizaciones a un índice, primero se ordena en el orden del índice; esta ordenación puede reducir de forma muy significativa el número de E/S aleatorias necesarias para actualizar los índices.

Estas optimizaciones de actualizaciones se encuentran implementadas en la mayoría de los sistemas de bases de datos. Consulte las notas bibliográficas para más información.

13.6.4. Optimización de consultas múltiples y exploraciones compartidas

Cuando se envía a ejecutar una secuencia de consultar, un optimizador de consultas puede aprovechar las expresiones comunes entre distintas consultas, evaluándolas una vez y reutilizándolas cuando se necesiten. Las consultas complejas pueden, de hecho, tener subexpresiones repetidas en distintas partes de la consulta, lo que también se puede aprovechar para reducir el coste de evaluación. Esta optimización se denomina **optimización de consultas múltiples.**

La **eliminación de subexpresiones comunes** optimiza las subexpresiones compartidas por distintas expresiones de un programa, calculando y guardando el resultado y volviéndolo a utilizar siempre que se encuentre de nuevo la subexpresión. La eliminación de subexpresiones comunes es una optimización estándar que se aplica en expresiones aritméticas en los compiladores de los lenguajes de programación. Aprovechar las subexpresiones comunes entre los planes de evaluación elegidos para cada secuencia de consultas es igual de útil en la evaluación de consultas de una base de datos y está implementado en algunas de ellas. Sin embargo, la optimización de consultas múltiples puede hacerse incluso mejor en algunos casos. Una consulta normalmente tiene más de un plan de evaluación, y una buena elección del conjunto de planes de evaluación de consultas para la secuencia de las consultas puede proporcionar mayor compartición y menor coste que el que se consigue eligiendo el plan de evaluación de menor coste para cada una de las consultas. Puede encontrar más detalles sobre la optimización de consultas múltiples en las referencias citadas en las notas bibliográficas.

La compartición de las exploraciones de relaciones entre varias consultas es otra forma limitada de optimización de consultas múltiples que implementan algunas bases de datos. La optimización de **exploración compartida** funciona de la siguiente forma: en lugar de leer la relación repetidas veces del disco, una por cada consulta que lo necesite, los datos se leen una sola vez del disco y se encauzan a cada una de las consultas. La optimización de exploración compartida resulta especialmente útil cuando varias consultas realizan una exploración de una única relación muy grande (normalmente una «tabla de datos»).

13.6.5. Optimización paramétrica de consultas

El caché de planes, visto en la Sección 13.4.3, se usa como heurística en muchas bases de datos. Recuerde que con el caché de planes, si una consulta se invoca con algunas constantes, el plan elegido por el optimizador se guarda en la caché y se reutiliza si la consulta se vuelve a enviar, incluso aunque las constantes de la consulta sean diferentes. Por ejemplo, suponga una consulta que tiene el nombre de departamento como parámetro, y devuelve todas las asignaturas del departamento. Con el caché de planes, el plan elegido cuando la consulta se ejecuta por primera vez, suponga que es el departamento de Música, se vuelve a utilizar si la consulta se ejecuta para cualquier otro departamento.

Esta reutilización de planes mediante el caché de planes es razonable si el plan de consulta óptimo no se ve afectado de forma significativa por los valores concretos de las constantes de la consulta. Sin embargo, si el plan se ve afectado por los valores de las constantes, la optimización paramétrica de consultas es una alternativa.

En la **optimización paramétrica de consultas,** una consulta se optimiza sin dar valores concretos a los parámetros, por ejemplo *nombre_dept* en el caso anterior. El optimizador genera varios planes, cada uno óptimo para distintos valores de los parámetros. El optimizador utilizaría un plan solo si es el óptimo para algunos posibles valores de los parámetros. El conjunto de planes alternativos que genera el optimizador se guardan. Cuando se realiza una consulta con valores concretos de sus parámetros, en lugar de realizar una optimización completa, se usa el plan de menor coste del conjunto de planes alternativos calculados anteriormente. Encontrar el de menor coste entre dichos planes normalmente requiere mucho menos tiempo que una optimización nueva completa. Consulte las notas bibliográficas para más información sobre la optimización paramétrica de consultas.

13.7. Resumen

- Dada una consulta, suele haber gran variedad de métodos para calcular la respuesta. Es responsabilidad del sistema transformar la consulta tal y como la introdujo el usuario en una consulta equivalente que pueda calcularse de manera más eficiente. El proceso de búsqueda de una buena estrategia para el procesamiento de la consulta se denomina *optimización de consultas*.
- La evaluación de las consultas complejas implica muchos accesos a disco. Dado que la transferencia de los datos desde el disco resulta lenta en comparación con la velocidad de la memoria principal y de la CPU del sistema informático, merece la pena asignar una cantidad considerable de procesamiento a la elección de un método que minimice los accesos al disco.
- Hay varias reglas de equivalencia que se pueden emplear para transformar una expresión en otra equivalente. Estas reglas se utilizan para generar de manera sistemática todas las expresiones equivalentes a la consulta dada.
- Cada expresión del álgebra relacional representa una secuencia concreta de operaciones. El primer paso para la selección de una estrategia de procesamiento de consultas es la búsqueda de una expresión del álgebra relacional que sea equivalente a la expresión dada y que se estime sea menos costosa de ejecutar.
- La estrategia que escoja el sistema de bases de datos para la evaluación de una operación depende del tamaño de cada relación y de la distribución de los valores dentro de las columnas. Para que puedan basar su elección de estrategia en información de confianza, los sistemas de bases de datos almacenan estadísticas para cada relación r. Entre estas estadísticas se encuentran:
  - El número de tuplas en la relación r.
  - El tamaño del registro (tupla) de la relación r en bytes.
  - El número de valores diferentes que aparecen en la relación r para un atributo determinado.
- La mayoría de los sistemas de bases de datos usan histogramas para almacenar el número de valores de un atributo en cada intervalo de valores. Los histogramas se suelen calcular utilizando el muestreo.
- Estas estadísticas permiten estimar el tamaño del resultado de varias operaciones, así como el coste de su ejecución. La información estadística sobre las relaciones resulta especialmente útil cuando se dispone de varios índices para ayudar al procesamiento de una consulta. La presencia de estas estructuras tiene una influencia significativa en la elección de una estrategia de procesamiento de consultas.
- Los planes alternativos de evaluación para cada expresión pueden generarse mediante reglas de equivalencia y se puede escoger el plan de menor coste para todas las expresiones. Se dispone de varias técnicas de optimización para reducir el número de expresiones y planes alternativos que hace falta generar.
- Se usan heurísticas para reducir el número de planes considerados y, por tanto, para reducir el coste de la optimización. Entre las reglas heurísticas para transformar las consultas del álgebra relacional están «Llevar a cabo las operaciones de selección tan pronto como sea posible», «Llevar a cabo las proyecciones tan pronto como sea posible» y «Evitar los productos cartesianos».
- Las vistas materializadas pueden usarse para acelerar el procesamiento de las consultas. La conservación incremental de las vistas es necesaria para actualizar de forma eficiente las vistas materializadas cuando se modifican las relaciones subyacentes. El diferencial de cada operación puede calcularse mediante expresiones algebraicas que impliquen a los diferenciales de las entradas de la operación. Entre otros aspectos relacionados con las vistas materializadas están el modo de optimizar las consultas haciendo uso de las vistas materializadas disponibles y el modo de seleccionar las vistas que hay que materializar.
- Se han propuesto distintas técnicas avanzadas de optimización, como la optimización de primeros-K, la minimización de reunión, la optimización de actualizaciones, la optimización de consultas múltiples y la optimización paramétrica de consultas.

Términos de repaso

- Optimización de consultas.
- Transformación de expresiones.
- Equivalencia de expresiones.
- Reglas de equivalencia.
  - Conmutatividad de la reunión.
  - Asociatividad de la reunión.
- Conjunto mínimo de reglas de equivalencia.
- Enumeración de las expresiones equivalentes.
- Estimación de la estadística.
- Información de catálogo.
- Estimación del tamaño.
  - Selección.
  - Selectividad.
  - Reunión.
- Histogramas.
- Estimación de los valores distintos.
- Muestras aleatorias.
- Elección de los planes de evaluación.

- Interacción de las técnicas de evaluación.
- Optimización basada en el coste.
- Optimización del orden de reunión.
  - Algoritmo de programación dinámica.
  - Orden de reunión en profundidad por la izquierda.
  - Orden interesante.
- Optimización heurística.
- Caché de planes.
- Elección del plan de acceso.
- Evaluación correlacionada.
- Descorrelación.
- Vistas materializadas.
- Mantenimiento de las vistas materializadas.
  - Recálculo.
  - Mantenimiento incremental.
  - Inserción.
  - Borrado.
  - Actualizaciones.
- Optimización de consultas con vistas materializadas.
- Selección de índices.
- Selección de vistas materializadas.
- Optimización de primeros-*K*.
- Minimización de reunión.
- Problema de Halloween.
- Optimización de consultas múltiples.

Ejercicios prácticos

13.1. Demuestre que se cumplen las siguientes equivalencias. Explique el modo en que se pueden aplicar para mejorar la eficiencia de determinadas consultas:

a. $E_1 \bowtie_\theta (E_2 - E_3) = (E_1 \bowtie_\theta E_2 - E_1 \bowtie_\theta E_3)$.

b. $\sigma_\theta({}_A\mathcal{G}_F(E)) = {}_A\mathcal{G}_F(\sigma_\theta(E))$, donde θ solo usa atributos de A.

c. $\sigma_\theta E_1 ⟕ E_2) = \sigma_\theta(E_1) ⟕ E_2$, donde θ solo usa atributos de E_1.

13.2. Para cada uno de los siguientes pares de expresiones, ponga ejemplos de relaciones que muestren que las expresiones no son equivalentes.

a. $\Pi_A(R - S)$ y $\Pi_A(R) - \Pi_A(S)$.

b. $\sigma_{B<4}({}_A\mathcal{G}_{máx\ (B)}(R))$ y ${}_A\mathcal{G}_{máx\ (B)}(\sigma_{B<4}(R))$.

c. En las expresiones anteriores, si las dos apariciones de *máx* se sustituyeran por *mín*, indique si las expresiones serían equivalentes.

d. $(R ⟕ S) ⟕ T$ and $R ⟕ (S ⟕ T)$

En otras palabras, la reunión externa por la izquierda no es asociativa. (Sugerencia: suponga que los esquemas de las tres relaciones son $R(a, b1)$, $S(a, b2)$ y $T(a, b3)$, respectivamente).

e. $\sigma_\theta(E_1 ⟕ E_2)$ y $E_1 ⟕ \sigma_\theta(E_2)$, donde θ solo usa atributos de E_2.

13.3. SQL permite las relaciones con duplicados (Capítulo 3).

a. Defina las versiones de las operaciones básicas del álgebra relacional σ, Π, ×, ⋈, −, ∪ y ∩ que se utilizan en relaciones con duplicados, de manera consistente con SQL.

b. Compruebe cuáles de las reglas de equivalencia de la 1 a la 7.b se cumplen para la versión multiconjunto del álgebra relacional definida en el apartado a.

13.4. Considere las relaciones $r_1(A, B, C)$, $r_2(C, D, E)$ y $r_3(E, F)$, con las claves principales A, C y E, respectivamente. Suponga que r_1 tiene 1.000 tuplas, r_2 tiene 1.500 tuplas y r_3 tiene 750 tuplas. Estime el tamaño de $r_1 \bowtie r_2 \bowtie r_3$ y diseñe una estrategia eficiente para el cálculo de la reunión.

13.5. Considere las relaciones $r_1(A, B, C)$, $r_2(C, D, E)$ y $r_3(E, F)$ del Ejercicio práctico 13.4. Suponga que no hay claves principales, excepto el esquema completo. Sean $V(C, r_1)$ 900, $V(C, r_2)$ 1.100, $V(E, r_2)$ 50 y $V(E, r_3)$ 100. Suponga que r_1 tiene 1.000 tuplas, r_2 tiene 1.500 tuplas y r_3 tiene 750 tuplas. Estime el tamaño de $r_1 \bowtie r_2 \bowtie r_3$ y diseñe una estrategia eficiente para el cálculo de la reunión.

13.6. Suponga que se dispone de un índice árbol B⁺ para *edificio* sobre la relación *departamento* y que no se dispone de ningún otro índice. Indique la mejor manera de tratar las siguientes selecciones que implican a la negación:

a. $\sigma_{\neg(edificio\ <\ \text{«Watson»})}(departamento)$

b. $\sigma_{\neg(edificio\ =\ \text{«Watson»})}(departamento)$

c. $\sigma_{\neg(edificio\ <\ \text{«Watson»}\ \vee\ presupuesto\ <\ 50000)}(departamento)$

13.7. Considere la consulta:

```
select *
from r, s
where upper(r.A) = upper(s.A);
```

donde «upper» es una función que devuelve su argumento convertido en mayúsculas.

a. Busque qué plan se genera de esta consulta en el sistema de bases de datos que utiliza.

b. Algunos sistemas de bases de datos usarían una reunión de bucle anidado para esta consulta, lo que puede ser muy ineficiente. Explique brevemente cómo se pueden usar la reunión asociativa y la reunión por mezcla con ella.

13.8. Indique las condiciones bajo las que las siguientes expresiones son equivalentes.

$${}_{A,B}\mathcal{G}_{agr(C)}(E_1 \bowtie E_2) \text{ y } ({}_A\mathcal{G}_{agr(C)}(E_1)) \bowtie E_2$$

donde *agr* indica cualquier operación de agregación. Indique cómo se pueden relajar las condiciones anteriores si *agr* es **mín** o **máx.**

13.9. Considere el tema de ordenaciones interesantes en optimización. Suponga que tiene una consulta que calcula el orden natural de un conjunto de relaciones *S*. Dado un subconjunto *S1* de *S*, indique cuáles son las ordenaciones interesantes de *S1*.

13.10. Demuestre que, con n relaciones, hay $(2(n-1))!/(n-1)!$ órdenes de reunión diferentes. Sugerencia: un **árbol binario completo** es aquel en el que cada nodo interno tiene exactamente dos hijos. Utilice el hecho de que el número de árboles binarios completos diferentes con n nodos hojas es:

$$\frac{1}{n}\binom{2(n-1)}{(n-1)}$$

Si se desea, se puede obtener la fórmula para el número de árboles binarios completos con n nodos a partir de la fórmula para el número de árboles binarios con n nodos. El número de árboles binarios con n nodos es:

$$\frac{1}{n+1}\binom{2n}{n}$$

Este número se conoce como **número de Catalan** y su obtención puede hallarse en cualquier libro de texto estándar sobre estructuras de datos o algoritmos.

13.11. Demuestre que el orden de reunión de menor coste puede calcularse en un tiempo $O(3^n)$. Suponga que se puede almacenar y examinar la información sobre un conjunto de relaciones (como el orden óptimo de reunión para el conjunto y el coste de ese orden de reunión) en un tiempo constante. (Si encuentra difícil este ejercicio, demuestre al menos la cota de tiempo menos estricta de $O(2^{2n})$).

13.12. Demuestre que, si solo se toman en consideración los árboles de reunión en profundidad por la izquierda, como en el optimizador System R, el tiempo empleado en buscar el orden de reunión más eficiente es del orden de $n2^n$. Suponga que solo hay un orden interesante.

13.13. Considere la base de datos de un banco de la Figura 13.9, en la que las claves primarias están subrayadas. Construya las siguientes consultas en SQL para esta base de datos relacional.

a. Escriba una consulta anidada en la relación *cuenta* para buscar para cada una de las sucursales cuyo nombre empiece por B, todas las cuentas con el saldo máximo de la sucursal.

b. Reescriba la consulta anterior, sin usar una subconsulta anidada; es decir, descorrelacione la consulta.

c. Indique un procedimiento (similar al que se describe en la Sección 13.4.4) para la descorrelación.

sucursal(*nombre_sucursal*, *ciudad_sucursal*, *activos*)
cliente(*nombre_cliente*, *calle_cliente*, *ciudad_cliente*)
préstamo(*número_préstamo*, *nombre_sucursal*, *cantidad*)
prestador(*nombre_cliente*, *número_préstamo*)
cuenta(*número_cuenta*, *nombre_sucursal*, *cantidad*)
depositante(*nombre_cliente*, *número_cuenta*)

Figura 13.9. Base de datos de un banco para el Ejercicio 13.13.

13.14. La versión de conjuntos del operador semirreunión $\ltimes$ se define de la siguiente forma:

$$r \ltimes_\theta s = \Pi_R (r \bowtie_\theta s)$$

donde R es el conjunto de atributos del esquema de r. La versión multiconjunto de la operación semirreunión devuelve el mismo conjunto de tuplas, pero cada tupla tiene exactamente tantas copias como tiene en r.

Considere la consulta anidada que se vio en la Sección 13.4.4 que busca los nombres de todos los profesores que enseñaron algún curso en el año 2007. Escriba una consulta en álgebra relacional usando la operación semirreunión de multiconjuntos, asegurándose de que el número de duplicados de cada nombre es el mismo que en la consulta de SQL (la operación semirreunión se usa ampliamente para la descorrelación de consultas anidadas).

Ejercicios

13.15. Suponga que se dispone de un árbol B$^+$ para (*nombre_dept*, *edificio*) para la relación *departamento*. Indique la mejor manera de tratar la selección siguiente:

$$\sigma_{(edificio<\text{«Watson»}) \wedge (presupuesto<55000) \wedge (nombre_dept=\text{«Música»})}(departamento)$$

13.16. Indique el modo de obtener las equivalencias siguientes mediante una secuencia de transformaciones usando las reglas de equivalencia de la Sección 13.2.1.

a. $\sigma_{\theta_1 \wedge \theta_2 \wedge \theta_3}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(\sigma_{\theta_3}(E)))$

b. $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta_3} E_2) = \sigma_{\theta_1}(E_1 \bowtie_{\theta_3}(\sigma_{\theta_2}(E_2)))$, donde θ_2 solo implica atributos de E_2

13.17. Considere las dos expresiones $\sigma_\theta(E_1 ⟕ E_2)$ y $\sigma_\theta(E_1 \bowtie E_2)$.

a. Demuestre con un ejemplo que las dos expresiones no son equivalentes, en general.

b. Indique una condición simple sobre el predicado θ, que si se satisface asegura que las dos expresiones son equivalentes.

13.18. Se dice que un conjunto de reglas de equivalencia está *completo* si, siempre que dos expresiones sean equivalentes, se puede obtener una de la otra mediante la aplicación de una secuencia de las reglas de equivalencia. Indique si el conjunto de reglas de equivalencia que se consideró en la Sección 13.2.1 es completo. (Sugerencia: considere la equivalencia $\sigma_{3=5}(r) = \{\}$.

13.19. Describa el modo de usar un histograma para estimar el tamaño de una selección de la forma $\sigma_{A \le v}(r)$.

13.20. Suponga que dos relaciones r y s tienen histogramas sobre los atributos $r.A$ y $s.A$ respectivamente, pero con intervalos diferentes. Sugiera el modo de usar los histogramas para estimar el tamaño de $r \bowtie s$. (Sugerencia: divida más los intervalos de cada histograma).

13.21. Considere la consulta

```
select A, B
from r
where r.B < some (select B
                  from s
                  where s.A = r.A)
```

Indique cómo descorrelacionar la consulta anterior usando la versión multiconjunto de la operación

13.22. Describa el modo de conservar de manera incremental el resultado de las siguientes operaciones, tanto para inserciones como para borrados:

a. Unión y diferencia de conjuntos.

b. Reunión externa por la izquierda.

13.23. Indique un ejemplo de expresión que defina una vista materializada y dos situaciones (conjuntos de estadísticas para las relaciones de entrada y sus diferenciales) tales que la conservación incremental de la vista sea mejor que su recálculo en una de las situaciones y que el recálculo sea mejor en la otra.

13.24. Suponga que se desean obtener los resultados de $r \bowtie s$ ordenados según un atributo de r y solo se desean las primeras K respuestas para algún K relativamente pequeño. Indique una buena forma de evaluar la consulta.

a. Cuando la reunión es sobre una clave externa de r que referencia a s, donde el atributo de clave externa se declara que no es nulo.

b. Cuando la reunión no es sobre una clave externa.

13.25. Considere la relación $r(A, B, C)$ con un índice sobre el atributo A. Indique un ejemplo de una consulta que se pueda responder usando solo el índice, sin tener que examinar las tuplas de la relación. (Los planes de consulta que usan solo el índice sin acceder a la relación se denominan planes *solo de índices*).

13.26. Suponga que tiene una consulta de actualización U. Indique una condición suficiente sobre U que asegure que no se produce el problema de Halloween, independientemente del plan de ejecución elegido o los índices que existan.

Notas bibliográficas

El trabajo precursor de Selinger et ál. [1979] describe la selección del camino de acceso en el optimizador System R, que fue uno de los primeros optimizadores de consultas relacionales. El procesamiento de consultas en Starburst, que se describe en Haas et ál. [1989], es la base de la optimización de consultas en DB2 de IBM.

Graefe y McKenna [1993] describen Volcano, un optimizador de consultas basado en reglas de equivalencia que, junto a su sucesor Cascades (Graefe [1995]), es la base de la optimización de consultas en SQL Server de Microsoft.

La estimación de las estadísticas de los resultados de las consultas, como el tamaño del resultado, se abordan en Ioannidis y Poosala [1995], Poosala et ál. [1996] y Ganguly et ál. [1996], entre otros. Las distribuciones no uniformes de valores causan problemas para la estimación del tamaño y del coste de las consultas. Las técnicas de estimación del coste que usan histogramas de las distribuciones de los valores se han propuesto para abordar el problema. Ioannidis y Christodoulakis [1993], Ioannidis y Poosala [1995] y Poosala et ál. [1996] presentan los resultados en esta área. El uso del muestreo aleatorio para la generación de histogramas se conoce bien en estadística; los temas sobre la generación de histogramas en el contexto de las bases de datos se trata en Chaudhuri et ál. [1998].

Klug [1982] fue uno de los primeros trabajos sobre optimización de expresiones del álgebra relacional con funciones de agregación. Yan y Larson [1995] y Chaudhuri y Shim [1994] estudian la optimización de consultas con agregación. La optimización de las consultas que contienen reuniones externas se describe en Rosenthal y Reiner [1984], Galindo-Legaria y Rosenthal [1992] y Galindo-Legaria [1994]. La optimización primeros-K se estudia en Carey y Kossmann [1998] y Bruno et ál. [2002].

La optimización de las subconsultas anidadas se trata en Kim [1982], Ganski y Wong [1987], Dayal [1987], Seshadri et ál. [1996] y en Galindo-Legaria y Joshi [2001].

Blakeley et ál. [1986] describen las técnicas para la conservación de las vistas materializadas. La optimización de los planes de conservación de las vistas materializadas se describe en Vista [1998] y Mistry et ál. [2001]. La optimización de consultas en presencia de vistas materializadas se aborda en Chaudhuri et ál. [1995]. La selección de índices y la selección de vistas materializadas se abordan en Ross et ál. [1996], Chaudhuri y Narasayya [1997].

La optimización de consultas de primeros-K se estudia en Carey y Kossmann [1998] y Bruno et ál. [2002]. Una colección de técnicas para la minimización de reuniones se ha agrupado bajo el nombre *optimización plana* (*tableau optimization*). La noción de un tableau la introdujo Aho et ál. [1979b] y Aho et ál. [1979a] y fue extendida posteriormente por Sagiv y Yannakakis [1981].

Los algoritmos de optimización paramétrica de consultas fueron propuestos por Ioannidis et ál. [1992], Ganguly [1998] y Hulgeri y Sudarshan [2003]. Sellis [1988] realizó los primeros trabajos sobre optimización de consultas múltiples, mientras que Roy et ál. [2000] demostraron cómo integrar la optimización de consultas múltiples en un optimizador de consultas basado en Volcano.

Galindo-Legaria et ál. [2004] describen el procesamiento de consultas y su optimización en las actualizaciones de bases de datos, incluyendo la optimización del mantenimiento de índices, los planes de mantenimiento de vistas materializadas y la comprobación de restricciones de integridad, junto con técnicas para gestionar el problema de Halloween.

Parte 4
Gestión de transacciones

El término *transacción* hace referencia a un conjunto de operaciones que forman una única unidad lógica de trabajo. Por ejemplo, la transferencia de dinero de una cuenta a otra es una transacción que consta de dos actualizaciones, una para cada cuenta.

Resulta importante que, o bien se ejecuten completamente todas las acciones de una transacción, o bien, en caso de fallo, se deshagan los efectos parciales de cada transacción incompleta. Esta propiedad se denomina *atomicidad*. Además, una vez ejecutada con éxito una transacción, sus efectos deben persistir en la base de datos —un fallo en el sistema no debe tener como consecuencia que la base de datos descarte una transacción que se haya completado con éxito. Esta propiedad se denomina *durabilidad*.

En los sistemas de bases de datos en los que se ejecutan de manera concurrente varias transacciones, si no se controlan las actualizaciones de datos compartidos, existe la posibilidad de que las transacciones operen sobre estados intermedios inconsistentes creados por las actualizaciones de otras transacciones. Esta situación puede dar lugar a actualizaciones erróneas de los datos almacenados en la base de datos. Por tanto, los sistemas de bases de datos deben proporcionar los mecanismos para aislar las transacciones de otras transacciones que se ejecuten de manera concurrente. Esta propiedad se denomina *aislamiento*.

El Capítulo 14 describe con detalle el concepto de transacción, incluidas las propiedades de atomicidad, durabilidad, aislamiento y otras propiedades proporcionadas por la abstracción de las transacciones. En concreto, el capítulo precisa el concepto de aislamiento por medio de un concepto denominado secuencialidad.

El Capítulo 15 describe varias técnicas de control de concurrencia que ayudan a implementar la propiedad del aislamiento. El Capítulo 16 describe el componente de las bases de datos para la administración de las recuperaciones, que implementa las propiedades de atomicidad y de durabilidad.

Entendido como un todo, el componente de gestión de transacciones de un sistema de bases de datos permite a los desarrolladores de aplicaciones centrarse en la implementación de las transacciones individualmente, ignorando los aspectos de concurrencia y tolerancia a fallos.

Transacciones

14

A menudo, desde el punto de vista del usuario de una base de datos, se considera a un conjunto de varias operaciones sobre una base de datos como una única operación. Por ejemplo, una transferencia de fondos desde una cuenta corriente a una cuenta de ahorros es una operación simple desde el punto de vista del cliente; sin embargo, en el sistema de base de datos está compuesta internamente por varias operaciones. Evidentemente es esencial que tengan lugar todas las operaciones o que, en caso de fallo, ninguna de ellas se produzca. Sería inaceptable efectuar el cargo de la transferencia en la cuenta corriente y que no se abonase en la cuenta de ahorros.

Se denomina **transacción** a una colección de operaciones que forman una única unidad lógica de trabajo. Un sistema de base de datos debe asegurar que la ejecución de las transacciones se realice adecuadamente a pesar de la existencia de fallos; o se ejecuta la transacción completa o no se ejecuta en absoluto. Además, debe gestionar la ejecución concurrente de las transacciones evitando introducir inconsistencias. Volviendo al ejemplo de la transferencia de fondos, una transacción que calcule el saldo total del cliente podría ver el saldo de la cuenta corriente antes de que sea cargado por la transacción de la transferencia de fondos, así como el saldo de la cuenta de ahorros después del abono. Como resultado, se obtendría un resultado incorrecto.

Este capítulo es una introducción a los conceptos básicos en el procesamiento de transacciones. En los Capítulos 15 y 16 se incluyen más detalles sobre el procesamiento concurrente de transacciones y la recuperación de fallos. En el Capítulo 26 se tratan temas adicionales acerca del procesamiento de transacciones.

14.1. Concepto de transacción

Una **transacción** es una **unidad** de la ejecución de un programa que accede y posiblemente actualiza varios elementos de datos. Una transacción se inicia mediante la ejecución de un programa de usuario escrito en un lenguaje de manipulación de datos de alto nivel (normalmente SQL) o en un lenguaje de programación (por ejemplo, C++ o Java), con acceso a la base de datos incorporado en JDBC u ODBC. Una transacción está delimitada por sentencias (o llamadas a función) de la forma **begin transaction** (iniciar transacción) y **end transaction** (finalizar transacción). La transacción consiste en todas las operaciones que se ejecutan entre **begin transaction** y **end transaction.**

Esta colección de pasos debe aparecer ante el usuario como una única unidad indivisible. Como una transacción es indivisible, se ejecuta completamente o no se ejecuta nada. Por tanto, si una transacción comienza a ejecutarse pero, por cualquier razón, falla en su ejecución, cualquier cambio que la transacción haya podido realizar en la base de datos hay que deshacerlo. Este requisito se mantiene independientemente de que la propia transacción falle (por ejemplo, realiza una división por cero), el sistema operativo se caiga o la propia computadora deje de funcionar. Como se verá más adelante, es difícil asegurar este requisito, ya que algunos de los cambios en la base de datos puede que todavía se encuentren en variables de la memoria principal correspondientes a la transacción, mientras que puede que otras ya se hayan escrito en la base de datos y guardado en el disco. Esta propiedad de «todo o nada» se denomina **atomicidad.**

Más aún, como una transacción es una única unidad, sus acciones no se pueden ver separadas de otras operaciones de la base de datos ni como parte de la transacción. Aunque se desea dar la impresión de transacción al usuario, en realidad se sabe que es muy diferente. Incluso una única sentencia de SQL implica muchos accesos separados a la base de datos, y una transacción puede consistir en muchas sentencias de SQL. Por tanto, el sistema de base de datos debe tener un cuidado especial a la hora de asegurar que las transacciones funcionan correctamente sin la interferencia de otras sentencias de la base de datos que se ejecuten concurrentemente. Esta propiedad se denomina **aislamiento.**

Incluso aunque el sistema asegure la correcta ejecución de una transacción, sirve de poco si el sistema deja de funcionar y, como resultado, el sistema «olvida» la transacción. Las acciones de una transacción deben permanecer después de las caídas del sistema. Esta propiedad se denomina **durabilidad.**

Como consecuencia de las tres propiedades anteriores, las transacciones son una forma ideal de estructurar la interacción con una base de datos. Esto nos lleva a imponer un requisito sobre la propia transacción. Una transacción debe mantener la consistencia de la base de datos; si una transacción se ejecuta atómicamente en aislamiento comenzando con una base de datos consistente, la base de datos debe seguir siendo consistente al finalizar la transacción. Este requisito de consistencia va más allá de las restricciones de integridad que se han tratado anteriormente (como las restricciones de clave primaria, las restricciones de integridad, las restricciones de **check** y similares). Se espera que las transacciones vayan más allá asegurando la conservación de esas restricciones de consistencia que son muy complejas de indicar usando construcciones de SQL sobre la integridad. Cómo lograrlo es responsabilidad del programador que codifica la transacción. Esta propiedad se denomina **consistencia.**

Para precisar lo anterior de forma más concisa, se necesita que el sistema de base de datos mantenga las siguientes propiedades de las transacciones:

- **Atomicidad.** O bien todas las operaciones de la transacción se realizan adecuadamente en la base de datos o no lo hace ninguna de ellas.
- **Consistencia.** La ejecución aislada de la transacción (es decir, sin otra transacción que se ejecute concurrentemente) conserva la consistencia de la base de datos.

- **Aislamiento.** Aunque se ejecuten varias transacciones concurrentemente, el sistema garantiza que para cada par de transacciones T_i y T_j, se cumple que desde el punto de vista de T_i, o bien T_j ha terminado su ejecución antes de que comience T_i, o bien T_j ha comenzado su ejecución después de que termine T_i. De este modo, cada transacción ignora al resto de las transacciones que se ejecuten concurrentemente en el sistema.
- **Durabilidad**. Tras la finalización con éxito de una transacción, los cambios realizados en la base de datos permanecen, incluso aunque se produzcan fallos en el sistema.

Estas propiedades a menudo reciben el nombre de **propiedades ACID;** el acrónimo se obtiene de la primera letra de cada una de las cuatro propiedades en inglés (*Atomicity*, *Consistency*, *Isolation* y *Durability*, respectivamente).

Como se verá más adelante, asegurar la propiedad de aislamiento puede tener un impacto adverso significativo en el rendimiento del sistema. Por esta razón, algunas aplicaciones comprometen la propiedad de aislamiento. Se verán estos compromisos tras estudiar primero el cumplimiento estricto de las propiedades ACID.

14.2. Un modelo simple de transacciones

Como SQL es un lenguaje complejo y potente, se va a comenzar el estudio de las transacciones con un leguaje de base de datos simple que se centra en el movimiento de los datos del disco a la memoria principal y de la memoria principal al disco. Para ello, se van a ignorar las operaciones **insert** y **delete** de SQL y se retrasará su consideración hasta la Sección 15.8. Las únicas operaciones reales sobre los datos se van a restringir en nuestro lenguaje simple a las operaciones aritméticas. Más adelante se tratarán las transacciones de forma realista, en un contexto de SQL con un mayor conjunto de operaciones. Los elementos de datos de este modelo simplificado incluyen un único valor de dato (un número en los ejemplos). Cada elemento de datos se identifica con un nombre (normalmente una letra, como A, B, C, etc.).

Se va a ilustrar el concepto de transacción usando una aplicación bancaria simple que consiste en varias cuentas y un conjunto de transacciones que acceden a las mismas y las actualizan. Las transacciones acceden a los datos mediante las dos operaciones siguientes:

- leer(X), que transfiere el dato X de la base de datos a una variable llamada también X, en una memoria intermedia en la memoria principal perteneciente a la transacción que ejecuta la operación leer.
- escribir(X), que transfiere el valor de la variable X desde la memoria intermedia de la memoria principal que ejecuta la operación escribir al dato X en la base de datos.

Es importante conocer si un cambio en un elemento de dato aparece solo en la memoria principal o se ha escrito en la base de datos en el disco. En un sistema de base de datos real, la operación escribir no tiene por qué producir necesariamente una actualización de los datos en el disco; la operación escribir puede almacenarse temporalmente en alguna parte y ejecutarse al disco más tarde. Sin embargo, por el momento se supondrá que la operación escribir actualiza inmediatamente la base de datos. Se volverá a este tema en el Capítulo 16.

Sea T_i una transacción para transferir 50 € de la cuenta A a la cuenta B. Se puede definir esta transacción como:

T_i : leer(A);
$A := A - 50$;
escribir(A);
leer(B);
$B := B + 50$;
escribir (B).

Considere ahora cada una de las propiedades ACID. (Para facilitar la presentación, se consideran en distinto orden al indicado por A-C-I-D).

- **Consistencia:** en este caso, el requisito de consistencia es que la suma de A y B no sea alterada al ejecutar la transacción. Sin el requisito de consistencia, ¡la transacción podría crear o destruir dinero! Se puede comprobar fácilmente que si una base de datos es consistente antes de ejecutar una transacción, sigue siéndolo después de ejecutar dicha transacción.

  La responsabilidad de asegurar la consistencia de una transacción es del programador de la aplicación que codifica dicha transacción. La comprobación automática de las restricciones de integridad puede facilitar esta tarea, como se vio en la Sección 4.4.
- **Atomicidad:** suponga que justo antes de ejecutar la transacción T_i los valores de las cuentas A y B son de 1.000 € y 2.000 €, respectivamente. Suponga ahora que durante la ejecución de la transacción T_i se produce un fallo que impide que dicha transacción finalice con éxito su ejecución. Además, suponga que el fallo tiene lugar después de ejecutarse la operación escribir(A), pero antes de ejecutarse la operación escribir(B). En este caso, los valores de las cuentas A y B que se ven reflejados en la base de datos son 950 € y 2.000 €. El sistema ha destruido 50 € de la cuenta A como resultado de este fallo. En particular se puede ver que ya no se conserva la suma $A + B$.

  Así, como consecuencia del fallo, el estado del sistema deja de reflejar el estado real del mundo que se supone que modela la base de datos. Un estado así se denomina **estado inconsistente.** Hay que asegurarse de que estas inconsistencias no sean visibles en un sistema de base de datos. Observe, sin embargo, que un sistema puede en algún momento encontrarse en un estado inconsistente. Aunque la transacción T_i se ejecute por completo, existe un punto en el que el valor de la cuenta A es de 950 € y el de la cuenta B es de 2.000 €, lo cual constituye claramente un estado inconsistente. Este estado, sin embargo, se sustituye eventualmente por otro estado consistente en el que el valor de la cuenta A es de 950 € y el de la cuenta B es de 2.050 €. De este modo, si la transacción no empieza nunca o se garantiza que se complete, un estado inconsistente así no será visible excepto durante la ejecución de la transacción. Ésta es la razón de que aparezca el requisito de atomicidad. Si se cumple la propiedad de atomicidad, o todas las acciones de la transacción se ven reflejadas en la base de datos, o no se ve reflejada ninguna de ellas.

  La idea básica para asegurar la atomicidad es la siguiente. El sistema de base de datos mantiene un registro de los valores antiguos (en disco) de aquellos datos sobre los que una transacción realiza una escritura. Esta información se escribe en un archivo llamado el *log* (registro). Si la transacción no completa su ejecución, el sistema de base de datos recupera los antiguos valores del registro para que parezca que la transacción no se ha ejecutado nunca. Estas ideas se muestran más adelante, en la Sección 14.4. La responsabilidad de asegurar la atomicidad es del sistema de base de datos; en concreto, lo maneja un componente llamado **sistema de recuperación,** que se describe en detalle en el Capítulo 16.
- **Durabilidad:** una vez que se completa con éxito la ejecución de una transacción, y después de comunicar al usuario que inició la transacción que se ha realizado la transferencia de fondos, un fallo en el sistema no debe producir la pérdida de datos correspondientes a dicha transferencia. La propiedad de durabilidad asegura que, una vez que se completa con éxito una transacción, persisten todas las modificaciones realizadas en la base de datos, incluso si hay un fallo en el sistema después de completarse la ejecución de dicha transacción.

A partir de ahora se asume que un fallo en la computadora del sistema puede producir una pérdida de datos de la memoria principal, pero los datos almacenados en disco nunca se pierden. En el Capítulo 16 se trata la protección contra la pérdida de datos. Se puede garantizar la durabilidad si se asegura que:

1. Las modificaciones realizadas por la transacción se guardan en disco antes de que finalice la transacción.
2. La información de las modificaciones realizadas por la transacción guardada en disco es suficiente para permitir a la base de datos reconstruir dichas modificaciones cuando el sistema se reinicie después del fallo.

El **sistema de recuperación** de la base de datos, que se describe en el Capítulo 16, es el responsable de asegurar la durabilidad, además de la atomicidad.

- **Aislamiento:** incluso si se aseguran las propiedades de consistencia y de atomicidad para cada transacción, si varias transacciones se ejecutan concurrentemente, se pueden entrelazar sus operaciones de un modo no deseado, produciendo un estado inconsistente.

  Por ejemplo, como se ha visto antes, la base de datos es inconsistente temporalmente durante la ejecución de la transacción para transferir fondos de la cuenta A a la cuenta B, con el total deducido escrito ya en A y el total incrementado todavía sin escribir en B. Si una segunda transacción que se ejecuta concurrentemente lee A y B en este punto intermedio y calcula $A + B$, observará un valor inconsistente. Además, si esta segunda transacción realiza después modificaciones en A y B basándose en los valores leídos, la base de datos puede permanecer en un estado inconsistente aunque ambas transacciones terminen.

  Una solución para el problema de ejecutar transacciones concurrentemente es ejecutarlas secuencialmente; es decir, una tras otra. Sin embargo, la ejecución concurrente de transacciones produce notables beneficios en el rendimiento, como se verá en la Sección 14.5. Se han desarrollado otras soluciones que permiten la ejecución concurrente de varias transacciones.

  Los problemas que causa la ejecución concurrente de transacciones se muestran en la Sección 14.5. La propiedad de aislamiento asegura que el resultado obtenido al ejecutar concurrentemente las transacciones es un estado del sistema equivalente a uno obtenido al ejecutar una tras otra en algún orden. Los principios del aislamiento se presentarán más adelante en la Sección 14.6. La responsabilidad de asegurar la propiedad de aislamiento es de un componente del sistema de base de datos llamado **componente de control de concurrencia,** que se presenta en el Capítulo 15.

14.3. Estructura de almacenamiento

Para entender cómo asegurar las propiedades de atomicidad y durabilidad de una transacción hay que entender correctamente cómo se pueden guardar y acceder a los distintos elementos de datos de la base de datos.

En el Capítulo 10 se describieron los distintos medios de almacenamiento que se pueden distinguir por su velocidad relativa, capacidad, resistencia a fallos y clasificación como volátil o no volátil. Se revisan estos conceptos y se añade otra clase de almacenamiento denominada **almacenamiento estable.**

- **Almacenamiento volátil.** La información que se encuentra en almacenamiento volátil no sobrevive a las caídas del sistema. Ejemplos de este tipo de memoria son la memoria principal y la memoria caché. El acceso al almacenamiento volátil es extremadamente rápido, tanto por su velocidad de acceso como por su capacidad para acceder a cualquier elemento de datos directamente.
- **Almacenamiento no volátil.** La información que se encuentra en almacenamiento no volátil no sobrevive a la caída del sistema. Ejemplos de almacenamiento no volátil incluyen los dispositivos de almacenamiento secundario, como los discos magnéticos y el almacenamiento flash, utilizados para el almacenamiento en línea, y los dispositivos de almacenamiento terciario, como los medios ópticos y las cintas magnéticas utilizadas para el archivado. En el estado actual de la tecnología, el almacenamiento no volátil es más lento que el volátil, especialmente por el acceso aleatorio. Tanto los dispositivos de almacenamiento secundario como terciario son, sin embargo, susceptibles de fallos, lo que puede generar pérdidas de información.
- **Almacenamiento estable.** La información que se encuentra en almacenamiento estable *nunca* se pierde (lo de *nunca* hay que tomarlo con cuidado pues teóricamente no se puede garantizar *nunca*, por ejemplo es posible, aunque extremadamente improbable, que ¡un agujero negro pueda engullir la tierra y destruir permanentemente todos los datos!). Aunque teóricamente es imposible conseguir un almacenamiento estable, puede conseguirse algo muy aproximado mediante técnicas que logran que sea muy improbable perder los datos. Para la implementación del almacenamiento estable, se replica la información en varios medios de almacenamiento no volátil (normalmente discos) con modos de fallo independientes. Las actualizaciones hay que realizarlas con sumo cuidado para asegurar que un fallo durante una actualización en almacenamiento estable no genere una pérdida de información. En la Sección 16.2.1 se trata la implementación de almacenamiento estable.

Las distinciones entre los distintos tipos de almacenamiento pueden ser menos claras en la práctica que en esta presentación. Por ejemplo, algunos sistemas, por ejemplo algunos controladores RAID, proporcionan una batería de respaldo, de forma que parte de la memoria principal puede sobrevivir a la caída del sistema y a los fallos de alimentación.

Para que una transacción sea duradera, es necesario escribir sus cambios en almacenamiento estable. De forma similar, para que una transacción sea atómica hay que escribir un registro en almacenamiento estable antes de que se escriba en disco cualquier cambio a la base de datos. Claramente, el grado en el que el sistema asegura la durabilidad y la atomicidad depende de cómo de estable sea realmente la implementación del almacenamiento estable. En algunos casos se considera suficiente una copia única en el disco, pero para las aplicaciones cuyos datos son muy valiosos y cuyas transacciones son muy importantes se requieren varias copias o, en otras palabras, una mejor aproximación al concepto ideal de almacenamiento estable.

14.4. Atomicidad y durabilidad de las transacciones

Como se ha visto anteriormente, una transacción puede que no siempre termine su ejecución correctamente. Una transacción de este tipo se denomina **abortada.** Si se pretende asegurar la propiedad de atomicidad, una transacción abortada no debe tener ningún efecto sobre el estado de la base de datos. Así, cualquier cambio que haya realizado la transacción abortada sobre la base de datos debe deshacerse. Una vez que se han deshecho los cambios efectuados por la transacción abortada, se dice que la transacción está **retrocedida.** Forma parte de la responsabilidad del esquema de recuperaciones gestionar las transacciones abortadas. Normalmente se lleva a cabo manteniendo un **registro** (*log*). Cada modificación que realiza una transacción en la base de datos se registra primero en el registro. Se guarda el identificador de la transacción que realiza la modificación, el identificador del elemento de datos que se modifica y el valor antiguo (previo a la modificación) y el nuevo

(tras la modificación) del dato. Solo entonces se realiza la modificación propiamente dicha de la base de datos. El mantenimiento de un registro posibilita deshacer una modificación para asegurar la atomicidad y durabilidad en caso de que se produzca un fallo durante la ejecución de una transacción. En el Capítulo 16 se tratan los detalles de la recuperación basada en registros (*log*).

Una transacción que termina correctamente se dice que está **comprometida** (*committed*). Una transacción comprometida que haya hecho modificaciones transforma la base de datos llevándola a un nuevo estado consistente, que permanece incluso aunque se produzca un fallo en el sistema.

Cuando una transacción se ha comprometido no se pueden deshacer sus efectos abortándola. La única forma de deshacer los cambios de una transacción comprometida es ejecutando una **transacción compensadora.** Por ejemplo, si una transacción añade 20 € a una cuenta, la transacción compensadora debería restar 20 € de la cuenta. Sin embargo, no siempre se puede crear dicha transacción compensadora. Por tanto, se deja al usuario la responsabilidad de crear y ejecutar transacciones compensadoras, y no la gestiona el sistema de base de datos. En el Capítulo 26 se incluye un estudio de las transacciones compensadoras.

Es necesario precisar qué se entiende por *terminación correcta* de una transacción. Se establece por tanto un modelo abstracto simple de transacción. Una transacción debe estar en uno de los siguientes estados:

- **Activa,** el estado inicial; la transacción permanece en este estado durante su ejecución.
- **Parcialmente comprometida,** después de ejecutarse la última sentencia.
- **Fallida,** tras descubrir que no puede continuar la ejecución normal.
- **Abortada,** después del retroceso de la transacción y de haber restablecido la base de datos a su estado anterior al comienzo de la transacción.
- **Comprometida,** tras completarse correctamente.

El diagrama de estados correspondiente a una transacción se muestra en la Figura 14.1. Se dice que una transacción se ha comprometido solo si ha llegado al estado de comprometida. Análogamente, se dice que una transacción ha abortado solo si ha llegado al estado de abortada. Una transacción se dice que ha **terminado** si se ha comprometido o se ha abortado.

Una transacción comienza en estado de activa. Cuando acaba su última sentencia, pasa al estado de parcialmente comprometida. En este punto, la transacción ha terminado su ejecución, pero es posible que aún tenga que ser abortada, puesto que los datos actuales pueden estar todavía en la memoria principal y puede producirse un fallo en el hardware antes de que se complete correctamente.

El sistema de base de datos escribe en disco la información suficiente para que, incluso al producirse un fallo, puedan reproducirse los cambios hechos por la transacción al reiniciar el sistema tras el fallo. Cuando se termina de escribir esta información, la transacción pasa al estado de comprometida.

Como se ha mencionado anteriormente, se asume que los fallos no provocan pérdidas de datos en el disco. Las técnicas para tratar las pérdidas de datos en el disco se muestran en el Capítulo 16.

Una transacción llega a estado de fallida después de que el sistema determine que dicha transacción no puede continuar su ejecución normal (por ejemplo, a causa de errores del hardware o lógicos). Una transacción de este tipo se debe retroceder. Después pasa a estado de abortada. En este punto, el sistema tiene dos opciones:

- Puede **reiniciar** la transacción, pero solo si esta se ha abortado a causa de algún error de hardware o software no provocado por la lógica interna de la transacción. Una transacción reiniciada se considera una nueva transacción.
- Puede **cancelar** la transacción. Normalmente se hace si hay algún error interno lógico que solo se puede corregir escribiendo de nuevo el programa de aplicación, o debido a una entrada incorrecta o a que no se hayan encontrado los datos deseados en la base de datos.

Hay que tener cuidado cuando se trabaja con **escrituras externas observables,** como en una pantalla de un usuario o al enviar un correo electrónico. Cuando tiene lugar una escritura así, esta no puede borrarse, puesto que puede haber sido vista fuera del sistema de base de datos. La mayoría de los sistemas permiten que tales escrituras tengan lugar solo después de que la transacción llegue al estado de comprometida. Una manera de implementar dicho esquema es hacer que el sistema de base de datos almacene temporalmente cualquier valor asociado con estas escrituras externas en memoria no volátil, y realice las escrituras reales solo si la transacción llega al estado de comprometida. Si el sistema falla después de que la transacción llegue al estado de comprometida, pero antes de que finalicen las escrituras externas, el sistema de base de datos puede llevar a cabo dichas escrituras externas (usando los datos de la memoria no volátil) cuando el sistema se reinicie.

La gestión de las escrituras externas puede ser más complicada en ciertas situaciones. Por ejemplo, suponga que la acción externa es dispensar dinero en un cajero automático y el sistema falla justo antes de que se dispense el dinero (se supone que el dinero se dispensa atómicamente). No tiene sentido dispensar el dinero cuando se reinicie el sistema, ya que el usuario probablemente ya no esté en el cajero. En tal caso se necesita una transacción compensadora, como devolver el dinero a la cuenta del usuario.

Como otro ejemplo, suponga un usuario que realiza una reserva por la web. Es posible que el sistema de bases de datos o el servidor de aplicación se caigan justo tras comprometer la transacción de reserva. También es posible que la conexión de red con el usuario se pierda justo tras comprometer la transacción de reserva. En cualquier caso, incluso aunque la transacción se haya comprometido, no se ha realizado la escritura externa. Para manejar estas situaciones hay que diseñar la aplicación de forma que cuando el usuario se vuelva a conectar a la aplicación web pueda ver si la transacción se ha realizado o no.

Para algunas aplicaciones puede ser deseable permitir a las transacciones activas que muestren datos a los usuarios, particularmente para transacciones de larga duración que se ejecutan durante minutos u horas. Desafortunadamente no se puede permitir dicha salida de datos observables a no ser que se quiera arriesgar la atomicidad de la transacción. En el Capítulo 26 se describen modelos alternativos de transacciones que proporcionan transacciones interactivas de larga duración.

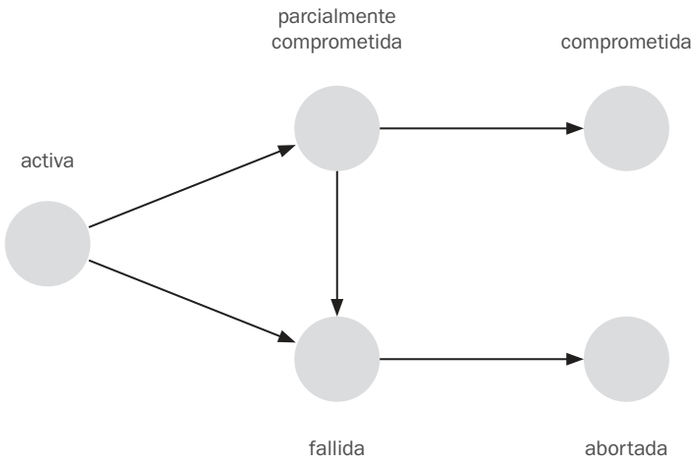


Figura 14.1. Diagrama de estados de una transacción.

14.5. Aislamiento de transacciones

Los sistemas de procesamiento de transacciones normalmente permiten la ejecución de varias transacciones concurrentemente. Permitir que varias transacciones actualicen concurrentemente los datos puede generar complicaciones en la consistencia de los mismos, como se ha visto. Asegurar la consistencia a pesar de la ejecución concurrente de las transacciones requiere un trabajo extra; es mucho más sencillo exigir que las transacciones se ejecuten **secuencialmente;** es decir, una a una, comenzado cada una solo después de que la anterior se haya completado. Sin embargo, existen dos buenas razones para permitir la concurrencia:

- **Mejora del rendimiento y uso de los recursos.** Una transacción consiste en varios pasos. Algunos implican operaciones de E/S; otros implican operaciones de la CPU. En una computadora la CPU y los discos pueden funcionar en paralelo. Por tanto, las operaciones de E/S se pueden realizar en paralelo con el procesamiento en la CPU. Se puede entonces explotar el paralelismo de la CPU y del sistema de E/S para ejecutar varias transacciones en paralelo. Mientras una transacción ejecuta una lectura o una escritura en un disco, otra puede ejecutarse en la CPU y otra tercera transacción ejecutar una lectura o una escritura en otro disco. Todo esto incrementa el **rendimiento** *(throughput)* del sistema, es decir, el número de transacciones que puede ejecutar en un tiempo dado. Análogamente, la **utilización** del procesador y del disco aumenta también; en otras palabras, el procesador y el disco están menos tiempo desocupados o sin hacer ningún trabajo útil.
- **Reducción del tiempo de espera.** Posiblemente se produzca una mezcla de tipos de transacciones que se ejecutan en el sistema, algunas cortas y otras largas. Si las transacciones se ejecutan secuencialmente, la transacción corta debe esperar a que la transacción larga anterior se complete, lo cual puede llevar a un retardo impredecible en la ejecución de la transacción. Si las transacciones operan sobre partes diferentes de la base de datos es mejor hacer que se ejecuten concurrentemente, compartiendo los ciclos de la CPU y los accesos a disco entre ambas. La ejecución concurrente reduce los retardos impredecibles en la ejecución de las transacciones. Además, se reduce también el **tiempo medio de respuesta:** el tiempo medio desde que una transacción comienza hasta que se completa.

La razón para usar la ejecución concurrente en una base de datos es esencialmente la misma que para usar **multiprogramación** en un sistema operativo.

Cuando se ejecutan varias transacciones concurrentemente, puede que no se cumpla la propiedad de aislamiento a pesar de que cada transacción individual sea correcta. En este apartado se presenta el concepto de planificaciones que ayudan a identificar aquellas ejecuciones que garantizan que se asegura la propiedad de aislamiento y, por tanto, la consistencia de la base de datos.

El sistema de base de datos debe controlar la interacción entre las transacciones concurrentes para evitar que se destruya la consistencia de la base de datos. Esto se lleva a cabo a través de una serie de mecanismos denominados **esquemas de control de concurrencia.** En el Capítulo 15 se estudian los esquemas de control de concurrencia; por ahora nos centraremos en el concepto de ejecución concurrente correcta.

Considere de nuevo el sistema bancario simplificado de la Sección 14.1, que tiene varias cuentas, y un conjunto de transacciones que acceden y modifican dichas cuentas. Sean T_1 y T_2 dos transacciones para transferir fondos de una cuenta a otra. La transacción T_1 transfiere 50 € de la cuenta A a la cuenta B y se define como sigue:

T_1 : leer(A);
$A := A - 50$;
escribir(A);
leer(B);
$B := B + 50$;
escribir(B).

La transacción T_2 transfiere el 10 por ciento del saldo de la cuenta A a la cuenta B, y se define como:

T_2: leer(A);
$temp := A * 0.1$;
$A := A - temp$;
escribir(A);
leer(B);
$B := B + temp$;
escribir(B).

Suponga que los valores actuales de las cuentas A y B son 1.000 € y 2.000 € respectivamente. Suponga que las dos transacciones se ejecutan de una en una en el orden T_1 seguida de T_2. Esta secuencia de ejecución se representa en la Figura 14.2 de la página siguiente. En esta figura la secuencia de pasos o instrucciones aparece en orden cronológico de arriba abajo, con las instrucciones de T_1 en la columna izquierda y las de T_2 en la derecha. Los valores finales de las cuentas A y B, después de que tenga lugar la ejecución de la Figura 14.2, son de 855 € y de 2.145 € respectivamente. De este modo, la suma total del saldo de las cuentas A y B, es decir, la suma $A + B$, se conserva tras la ejecución de ambas transacciones.

TENDENCIAS EN CONCURRENCIA

Diversas tendencias actuales en el campo de la computación conducen a un incremento en la concurrencia. Como los sistemas de bases de datos explotan esta concurrencia para aumentar el rendimiento global del sistema, habrá, necesariamente, un incremento en el número de transacciones concurrentes.

Las primeras computadoras solo tenían un procesador. Por tanto, no existía nunca concurrencia real en la computadora. La única concurrencia era la aparente que creaba el sistema operativo al compartir el procesador entre varios procesos distintos o tareas. En las computadoras modernas es probable disponer de varios procesadores. Pueden ser realmente procesadores distintos que formen parte de la computadora. Sin embargo, incluso un único procesador puede ser capaz de ejecutar más de un proceso a la vez teniendo varios *núcleos*. El procesador Intel Core Duo es un ejemplo muy conocido de procesador multinúcleo.

Para que los sistemas de bases de datos aprovechen la ventaja de disponer de varios procesadores y varios núcleos, se usan dos enfoques. Uno es encontrar el paralelismo en una única transacción o consulta. Otro consiste en permitir un gran número de transacciones concurrentes.

Muchos proveedores de servicio utilizan para ofrecer este grandes colecciones de computadoras en lugar de grandes computadoras centrales. Realizan esta elección por el menor coste que supone. Como resultado se produce un incremento en el grado de concurrencia de que se puede disponer.

En las notas bibliográficas se pueden encontrar referencias a textos que describen estos avances en arquitectura de computadoras y computación paralela. En el Capítulo 18 se describen los algoritmos para construir sistemas de bases de datos paralelos, que explotan múltiples procesadores y núcleos.

Análogamente, si las transacciones se ejecutan de una en una en el orden T_2 seguida de T_1, entonces la secuencia de ejecución es la de la Figura 14.3. De nuevo, como se esperaba, se conserva la suma $A + B$ y los valores finales de las cuentas A y B son de 850 € y de 2.150 € respectivamente.

Las secuencias de ejecución que se acaban de describir se denominan **planificaciones.** Representan el orden cronológico en el que se ejecutan las instrucciones en el sistema. Obviamente una planificación para un conjunto de transacciones debe consistir en todas las instrucciones de dichas transacciones, y debe conservar el orden en que aparecen las instrucciones en cada transacción individual. Por ejemplo, en la transacción T_1 la instrucción escribir(A) debe aparecer antes de la instrucción leer(B) en cualquier planificación válida. En la planificación se incluye la operación **commit** para indicar que la transacción ha entrado en el estado de comprometida. En los párrafos siguientes, planificación 1 se referirá a la primera secuencia de ejecución (T_1 seguida de T_2) y planificación 2 a la segunda secuencia de ejecución (T_2 seguida de T_1).

Estas planificaciones son **secuenciales.** Cada planificación secuencial consiste en una secuencia de instrucciones de varias transacciones, en la que las instrucciones pertenecientes a una única transacción están juntas en dicha planificación. Recordando una fórmula bien conocida de combinatoria, para un conjunto de n transacciones existen $n!$ planificaciones secuenciales válidas distintas.

Cuando el sistema de bases de datos ejecuta concurrentemente varias transacciones, la planificación correspondiente no tiene por qué ser secuencial. Si dos transacciones se ejecutan concurrentemente, el sistema operativo puede ejecutar una transacción durante un tiempo, luego realizar un cambio de contexto, ejecutar la segunda transacción durante cierto tiempo, cambiar de nuevo a la primera transacción durante un tiempo, y así sucesivamente. Si hay varias transacciones, todas ellas comparten el tiempo de la CPU.

Son posibles muchas secuencias de ejecución, puesto que se pueden intercalar varias instrucciones de ambas transacciones. En general, no es posible predecir exactamente cuántas instrucciones de una transacción se ejecutarán antes de que la CPU cambie a otra transacción.[1]

Volviendo al ejemplo anterior, suponga que las dos transacciones se ejecutan concurrentemente. Una posible planificación se muestra en la Figura 14.4. Una vez que la ejecución tiene lugar, se llega al mismo estado que cuando las transacciones se ejecutan secuencialmente en el orden T_1 seguida de T_2. La suma $A + B$ se conserva igualmente.

No todas las ejecuciones concurrentes producen un estado correcto. Como ejemplo, considere la planificación de la Figura 14.5. Después de ejecutarse esta planificación se llega a un estado cuyos valores finales de las cuentas A y B son de 950 € y 2.100 € respectivamente. Este estado final es un *estado inconsistente*, ya que se han ganado 50 € al procesar la ejecución concurrente. Realmente la ejecución de las dos transacciones no conserva la suma $A + B$.

Si se deja el control de la ejecución concurrente completamente al sistema operativo, son posibles muchas planificaciones, incluyendo las que dejan a la base de datos en un estado inconsistente, como la que se acaba de describir. Es una tarea del sistema de base de datos asegurar que cualquier planificación que se ejecute lleva a la base de datos a un estado consistente. El componente del sistema de base de datos que realiza esta tarea se denomina componente de **control de concurrencia.**

Se puede asegurar la consistencia de la base de datos en una ejecución concurrente si se está seguro de que cualquier planificación que se ejecute tiene el mismo efecto que otra que se hubiese ejecutado sin concurrencia. Es decir, la planificación debe ser, en cierto modo, equivalente a una planificación secuencial. Estas planificaciones se denominan planificaciones **serializables.**

T_1	T_2
leer(A)	
$A := A - 50$	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
commit	
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)
	commit

Figura 14.2. Planificación 1: planificación secuencial de T_1 seguida de T_2.

T_1	T_2
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)
	commit
leer(A)	
$A := A - 50$	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
commit	

Figura 14.3. Planificación 2: planificación secuencial de T_2 seguida de T_1.

T_1	T_2
leer(A)	
$A := A - 50$	
escribir(A)	
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
leer(B)	
$B := B + 50$	
escribir(B)	
commit	
	leer(B)
	$B := B + temp$
	escribir(B)
	commit

Figura 14.4. Planificación 3: planificación concurrente equivalente a la planificación 1.

1 El número de planificaciones posibles para un conjunto de n transacciones es muy grande. Existen $n!$ planificaciones serie diferentes. Considerando todas las posibles formas en que se pueden entrelazar los pasos de las transacciones, el número total de posibles planificaciones es mucho mayor que $n!$

14.6. Secuencialidad

Antes de considerar cómo el componente de control de concurrencia del sistema de bases de datos puede asegurar la secuencialidad, vamos a considerar cómo determinar cuándo una planificación es secuenciable. Ciertamente, las planificaciones serie son secuenciables, pero si se entrelazan los pasos de varias transacciones, resulta difícil determinar si una planificación es secuenciable. Puesto que las transacciones son programas, es difícil calcular cuáles son las operaciones exactas que realiza una transacción y cómo interaccionan las operaciones de varias transacciones. Por este motivo no se van a considerar los distintos tipos de operaciones que puede realizar una transacción sobre un elemento de datos, sino solo dos operaciones: leer y escribir. Se supone que entre una instrucción leer(Q) y otra escribir(Q) sobre un elemento de datos Q, una transacción puede realizar una secuencia arbitraria de operaciones con la copia Q que reside en la memoria intermedia local de dicha transacción. De este modo las únicas operaciones significativas de la transacción son, desde el punto de vista de la planificación, las instrucciones leer y escribir. Las operaciones commit, aunque relevantes, no se consideran hasta la Sección 14.7. Por tanto, se pueden mostrar solo las instrucciones leer y escribir en las planificaciones, como se muestra en la planificación 3 de la Figura 14.6.

En esta sección se estudian diferentes formas de equivalencia de planificación; pero se centra en una forma particular denominada **secuencialidad en cuanto a conflictos.**

Considere una planificación S en la cual hay dos instrucciones consecutivas I y J, pertenecientes a las transacciones T_i y T_j respectivamente ($i \neq j$). Si I y J se refieren a distintos elementos de datos se pueden intercambiar sin afectar al resultado de cualquier instrucción de la planificación. Sin embargo, si I y J se refieren al mismo elemento Q, entonces el orden de los dos pasos puede ser importante. Puesto que solo se tienen en cuenta las instrucciones leer y escribir, se deben considerar cuatro casos:

1. $I =$ leer(Q), $J =$ leer(Q). El orden de I y J no importa, puesto que T_i y T_j leen el mismo valor de Q, independientemente del orden.
2. $I =$ leer(Q), $J =$ escribir(Q). Si I está antes que J, entonces T_i no lee el valor de Q que escribe la instrucción J de T_j. Si J está antes que I, entonces T_i lee el valor de Q escrito por T_j. Por tanto, el orden de I y J es importante.
3. $I =$ escribir(Q), $J =$ leer(Q). El orden de I y J es importante por razones similares a las del caso anterior.
4. $I =$ escribir(Q), $J =$ escribir(Q). Puesto que ambas instrucciones son operaciones escribir, el orden de dichas instrucciones no afecta ni a T_i ni a T_j. Sin embargo, el valor que obtendrá la siguiente instrucción leer(Q) de S sí se ve afectado, ya que únicamente se conserva en la base de datos la última de las dos instrucciones escribir. Si no hay ninguna otra instrucción escribir(Q) después de I y J en S, entonces el orden de I y J afecta directamente al valor final de Q en el estado de la base de datos que se obtiene con la planificación S.

De esta manera, solo en el caso en el cual I y J son instrucciones leer no tiene importancia el orden de ejecución de las mismas.

Se dice que I y J están en **conflicto** si existen operaciones de diferentes transacciones sobre el mismo elemento de datos, y al menos una de esas instrucciones es una operación escribir.

Para ilustrar el concepto de instrucciones conflictivas considere la planificación 3 mostrada en la Figura 14.5. La instrucción escribir(A) de T_1 está en conflicto con la instrucción leer(A) de T_2. Sin embargo, la instrucción escribir(A) de T_2 no está en conflicto con la instrucción leer(B) de T_1, ya que las dos instrucciones acceden a diferentes elementos de datos.

T_1	T_2
leer(A)	
$A := A - 50$	
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
	leer(B)
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
commit	
	$B := B + temp$
	escribir(B)
	commit

Figura 14.5. Planificación 4: planificación concurrente que genera un estado inconsistente.

T_1	T_2
leer(A)	
escribir(A)	
	leer(A)
	escribir(A)
leer(B)	
escribir(B)	
	leer(B)
	escribir(B)

Figura 14.6. Planificación 3: solo se muestran las instrucciones leer y escribir.

T_1	T_2
leer(A)	
escribir(A)	
	leer(A)
leer(B)	
	escribir(A)
escribir(B)	
	leer(B)
	escribir(B)

Figura 14.7. Planificación 5: planificación 3 tras intercambiar un par de instrucciones.

T_1	T_2
leer(A)	
escribir(A)	
leer(B)	
escribir(B)	
	leer(A)
	escribir(A)
	leer(B)
	escribir(B)

Figura 14.8. Planificación 6: planificación secuencial equivalente a la planificación 3.

T_3	T_4
leer(Q)	
	escribir(Q)
escribir(Q)	

Figura 14.9. Planificación 7.

Sean I y J instrucciones consecutivas de una planificación S. Si I y J son instrucciones de transacciones diferentes y además no están en conflicto, entonces se puede cambiar el orden de I y J para obtener una nueva planificación S'. S es equivalente a S', ya que todas las instrucciones aparecen en el mismo orden en ambas planificaciones, salvo I y J , cuyo orden no es importante.

Como la instrucción escribir(A) de T_2 en la planificación 3 de la Figura 14.6 no está en conflicto con la instrucción leer(B) de T_1, se pueden intercambiar dichas instrucciones para generar una planificación equivalente, la planificación 5, que se muestra en la Figura 14.7. Independientemente de cuál sea el estado inicial del sistema, las planificaciones 3 y 5 producen el mismo estado final del sistema.

Se puede continuar intercambiando instrucciones no conflictivas como sigue:

- Intercambiar la instrucción leer(B) de T_1 con la instrucción leer(A) de T_2 .
- Intercambiar la instrucción escribir(B) de T_1 con la instrucción escribir(A) de T_2 .
- Intercambiar la instrucción escribir(B) de T_1 con la instrucción leer(A) de T_2 .

El resultado final de estos intercambios, la planificación 6 de la Figura 14.8, es una planificación secuencial. Fíjese que la planificación 6 es exactamente la misma que la planificación 1, pero solo muestra las instrucciones leer y escribir. Por tanto, se demuestra que la planificación 3 es equivalente a una planificación secuencial. Esta equivalencia implica que independientemente del estado inicial, la planificación 3 produce el mismo estado final que una planificación secuencial.

Si una planificación S se puede transformar en otra S' mediante una serie de intercambios de instrucciones no conflictivas, se dice que S y S' son **equivalentes en cuanto a conflictos.**[2]

No todas las planificaciones en serie son equivalentes entre sí en cuanto a conflictos. Por ejemplo, las planificaciones 1 y 2 no son equivalentes en cuanto a conflictos.

El concepto de equivalencia en cuanto a conflictos lleva al concepto de secuencialidad en cuanto a conflictos. Se dice que una planificación S es **secuenciable en cuanto a conflictos** si es equivalente en cuanto a conflictos a una planificación secuencial. Así, la planificación 3 es secuenciable en cuanto a conflictos, ya que es equivalente en cuanto a conflictos a la planificación secuencial 1.

Finalmente, considere la planificación 7 de la Figura 14.9; solo consta de las operaciones significativas (es decir leer y escribir) de las transacciones T_3 y T_4. Esta planificación no es secuenciable en cuanto a conflictos, ya que no es equivalente ni a la planificación secuencial $<T_3,T_4>$ ni a $<T_4,T_3>$.

A continuación se presenta un método simple y eficiente para determinar la secuencialidad en cuanto a conflictos de una planificación. Considere una planificación S. Se construye un grafo, llamado **grafo de precedencia,** de S. Este grafo se define como $G = (V, E)$, donde V es el conjunto de vértices y E es un conjunto de aristas. El conjunto de vértices es el conjunto de las transacciones de la planificación. El conjunto de vértices son todas las secuencias $T_i \rightarrow T_j$ para las que se cumplen tres condiciones:

1. T_i ejecuta escribir(Q) antes de que T_j ejecute leer(Q).

2. T_i ejecuta leer(Q) antes de que T_j ejecute escribir(Q).

3. T_i ejecuta escribir(Q) antes de que T_j ejecute escribir(Q).

Si existe una arista $T_i \rightarrow T_j$ en el grafo de precedencia, entonces en cualquier planificación S' equivalente a S, T_i debe aparecer antes que T_j.

2 Se usa el término *equivalente en cuanto a conflictos* para distinguir la forma en que se ha definido la equivalencia de otras definiciones que se tratarán más adelante en este capítulo.

Por ejemplo, el grafo de precedencia de la planificación 1 en la Figura 14.10a contiene una sola arista $T_1 \rightarrow T_2$, ya que todas las instrucciones de T_1 se ejecutan antes de que se ejecute la primera instrucción de T_2. De forma similar, en la Figura 14.10b se muestra el grafo de precedencia de la planificación 2 con una única arista $T_2 \rightarrow T_1$, ya que todas las instrucciones de T_2 se ejecutan antes de que se ejecute la primera instrucción de T_1.

El grafo de precedencia de la planificación 4 se muestra en la Figura 14.11. Contiene la arista $T_1 \rightarrow T_2$, ya que T_1 ejecuta leer(A) antes de que T_2 ejecute escribir(A). También contiene la arista $T_2 \rightarrow T_1$, ya que T_2 ejecuta leer(B) antes de que T_1 ejecute escribir(B).

Si el grafo de precedencia de S tiene un ciclo, entonces la planificación S no es secuenciable en cuanto a conflictos. Si el grafo no contiene ciclos, entonces la planificación S es secuenciable en cuanto a conflictos.

Un **orden de secuencialidad** de una transacción se puede obtener como un orden lineal formado por el orden parcial del grafo anterior. Este proceso se denomina **ordenación topológica**. En general, existen varias ordenaciones lineales posibles que se pueden obtener de una ordenación topológica. Por ejemplo, el grafo de la Figura 14.12a tiene las dos ordenaciones topológicas aceptables que se muestran en las Figuras 14.12b y 14.12c.

Por tanto, para comprobar la secuencialidad en cuanto a conflictos, hay que construir el grafo de precedencia y utilizar el algoritmo de detección de ciclos. Este algoritmo se puede encontrar en los libros habituales de algoritmos. Algoritmos de detección de ciclos, como los que se basan en una búsqueda de primero en profundidad, requieren un orden de operaciones de n_2, donde n es el número de vértices del grafo (es decir, el número de transacciones).

Figura 14.10. Grafo de precedencia de (a) planificación 1 y (b) planificación 2.

Figura 14.11. Grafo de precedencia para la planificación 4.

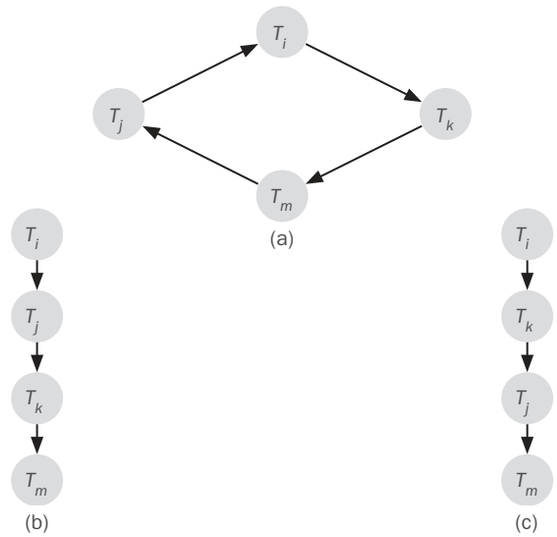


Figura 14.12. Ilustración de ordenación topológica.

Volviendo a nuestros ejemplos anteriores, fíjese que los grafos de precedencia de las planificaciones 1 y 2 (Figura 14.10) de hecho no contienen ciclos. El grafo de precedencia de la planificación 4 (Figura 14.11), por otra parte, contiene un ciclo, lo que indica que esta planificación no es secuenciable en cuanto a conflictos.

Es posible encontrar dos planificaciones que produzcan el mismo resultado y que no sean equivalentes en cuanto a conflictos. Por ejemplo, considere la transacción T_5, que transfiere 10 € de la cuenta B a la A. Sea la planificación 8 que se define en la Figura 14.13. Se puede afirmar que la planificación 8 no es equivalente en cuanto a conflictos a la planificación secuencial $<T_1,T_5>$, ya que en la planificación 8 la instrucción escribir(B) de T_5 está en conflicto con la instrucción leer(B) de T_1. Esto crea una arista $T_5 \rightarrow T_1$ en el grafo de precedencia. De forma similar, se puede observar que la instrucción escribir(A) de T_1 está en conflicto con la instrucción leer de T_5, creando una arista $<T_5 \rightarrow T_1>$. De esta forma se demuestra que el grafo de precedencia tiene un ciclo y que la planificación 8 no es secuenciable. Sin embargo, los valores finales de las cuentas A y B son los mismos después de ejecutar tanto la planificación 8 como la planificación secuencial $<T_1,T_5>$: 960 € y 2.040 €, respectivamente.

Con este ejemplo se puede observar que existen definiciones de equivalencia de planificaciones que son menos rigurosas que la de equivalencia en cuanto a conflictos. Para que el sistema pueda determinar que el resultado de la planificación 8 sea el mismo que el de la planificación secuencial $<T_1,T_5>$, debe analizar los cálculos realizados por T_1 y T_5, en lugar de tener solo en cuenta las operaciones leer y escribir. En general, tal análisis es difícil de implementar y es costoso en términos de cómputo. En nuestro ejemplo, el resultado final es el mismo que el de una planificación secuencial por el hecho matemático de que la suma y la resta son conmutativas. Mientras que esto resulta fácil de ver en este ejemplo sencillo, en el caso general no es tan sencillo pues una transacción puede ser una compleja sentencia de SQL, un programa con llamadas JDBC, etc.

Sin embargo, existen otras definiciones de equivalencia de planificación que se basan únicamente en las operaciones leer y escribir. Una de estas definiciones es la *equivalencia en cuanto a vistas*, una definición que lleva al concepto de *secuencialidad en cuanto a vistas*. La secuencialidad en cuanto a vistas no se usa en la práctica debido a su alto grado de complejidad computacional.[3] La secuencialidad en cuanto a vistas se verá en el Capítulo 15, pero por completitud, fíjese que el ejemplo de la planificación 8 no es secuenciable en cuanto a vistas.

T_1	T_5
leer(A)	
$A := A - 50$	
escribir(A)	
leer(B)	
	$B := B + 50$
	escribir(B)
leer(B)	
$B := B - 10$	
escribir(B)	
	leer(A)
	$A := A + 10$
	escribir(A)

Figura 14.13. Planificación 8.

3 La comprobación de secuencialidad en cuanto a vistas ha demostrado ser un problema NP-completo, lo que significa que es prácticamente cierto que no existe ningún algoritmo eficiente para tal comprobación.

14.7. Aislamiento y atomicidad de transacciones

Hasta este momento se han estudiado las planificaciones asumiendo implícitamente que no había fallos en las transacciones. Ahora se va a estudiar el efecto de los fallos en una transacción durante una ejecución concurrente.

Si la transacción T_i falla, por la razón que sea, es necesario deshacer el efecto de dicha transacción para asegurar la propiedad de atomicidad de la misma. En un sistema que permita la concurrencia es necesario asegurar también que toda transacción T_j que dependa de T_i (es decir, T_j lee datos que ha escrito T_i) se aborta. Para lograr esta garantía es necesario poner restricciones al tipo de planificaciones permitidas en el sistema.

En las dos subsecciones siguientes se estudian las planificaciones que son aceptables desde el punto de vista de la recuperación del fallo de una transacción. En el Capítulo 15 se describe la manera de asegurar que solo se generan dichas planificaciones aceptables.

14.7.1. Planificaciones recuperables

Considere la planificación 9 de la Figura 14.14, en la cual la transacción T_7 solo realiza una instrucción: leer(A). Esta se denomina **planificación parcial,** pues no se ha incluido ninguna operación **commit** o **abort** para T_6. Como puede verse, el sistema permite que T_7 se complete inmediatamente después de ejecutar la instrucción leer(A). Así se completa T_7 mientras que T_6 está todavía en estado activo. Suponga ahora que T_6 falla antes de completarse. T_7 ha leído el valor del elemento de datos A que escribió T_6. Por tanto, se dice que T_7 es **dependiente** de T_6. Por ello, se debe abortar T_7 para asegurar la atomicidad de la transacción. Sin embargo, T_7 ya se ha comprometido y no se puede abortar. De este modo se llega a una situación en la cual es imposible recuperarse correctamente del fallo de T_6.

La planificación 9 es un ejemplo de planificación *no recuperable*. Una **planificación recuperable** es aquella en la que para todo par de transacciones T_i y T_j tales que T_j lee elementos de datos que ha escrito previamente T_i, la operación comprometer de T_i aparece antes que la de T_j. Para que el ejemplo de la planificación 9 fuese recuperable, T_7 debería retrasar el compromiso hasta después de que T_6 comprometiese.

T_6	T_7
leer(A)	
escribir(A)	
	leer(A)
	commit
leer(B)	

Figura 14.14. Planificación 9, una planificación no recuperable.

14.7.2. Planificaciones sin cascada

Incluso si una planificación es recuperable, hay que retroceder varias transacciones para recuperar correctamente el estado previo a un fallo en una transacción T_i. Estas situaciones se producen si las transacciones leen datos que ha escrito T_i. Como ejemplo, considere la planificación parcial de la Figura 14.15. La transacción T_8 escribe un valor de A que lee la transacción T_9. La transacción T_9 escribe un valor de A que lee la transacción T_{10}. Suponga que en ese momento falla T_8. Se debe retroceder T_8. Puesto que T_9 depende de T_8, se debe retroceder T_9. Puesto que T_{10} depende de T_9, se debe retroceder T_{10}. Este fenómeno en el cual un fallo en una única transacción provoca una serie de retrocesos de transacciones se denomina **retroceso en cascada.**

No es deseable el retroceso en cascada, ya que provoca un aumento significativo del trabajo necesario para deshacer cálculos. Es conveniente restringir las planificaciones a aquellas en las que no puedan producirse retrocesos en cascada. Tales planificaciones se denominan planificaciones *sin cascada*. Formalmente, una **planificación sin cascada** es aquella en la que, para todo par de transacciones T_i y T_j tales que T_j lee un elemento de datos que ha escrito previamente T_i, la operación comprometer de T_i aparece antes que la operación de lectura de T_j. Es sencillo comprobar que toda planificación sin cascada es también recuperable.

T_8	T_9	T_{10}
leer(A)		
leer(B)		
escribir(A)		
	leer(A)	
	escribir(A)	
		leer(A)
abort		

Figura 14.15. Planificación 10.

14.8. Niveles de aislamiento de transacciones

La secuencialidad es un concepto útil que permite a los programadores ignorar los problemas de concurrencia cuando codifican transacciones. Si todas las transacciones poseen la propiedad de mantener la consistencia de la base de datos si se ejecutan solas, entonces la secuencialidad asegura que la ejecución concurrente mantiene la consistencia. Sin embargo, los protocolos requeridos para asegurar la secuencialidad tal vez permitan una concurrencia muy baja para ciertas aplicaciones. En estos casos se usan niveles de consistencia más débiles. El uso de estos niveles de consistencia más débiles traslada los problemas a los programadores, que tienen que asegurar la corrección de la base de datos.

La norma SQL también permite que una transacción especifique si se puede ejecutar de forma que no sea secuenciable con respecto al resto de transacciones. Por ejemplo, una transacción puede trabajar en el nivel de aislamiento **lectura no comprometida,** que permite que la transacción lea un elemento de datos aunque lo haya escrito una transacción que no está comprometida. SQL proporciona esta funcionalidad para su uso en grandes transacciones cuyo resultado no tiene que ser preciso. Si estas transacciones tuviesen que ejecutarse de forma secuenciada, podrían interferir con otras transacciones, lo que generaría retrasos en la ejecución.

Los niveles de aislamiento que especifica la norma SQL son los siguientes:

- **Secuenciable:** normalmente asegura la ejecución secuencial. Sin embargo, como se tratará en breve, algunos sistemas de bases de datos implantan este nivel de aislamiento de una forma que puede, en algunos casos, permitir ejecuciones no secuenciales.
- **Lectura repetible:** permite que solo se puedan leer datos comprometidos y, además, requiere que entre dos lecturas de un dato en una transacción, ninguna otra transacción pueda actualizarlo. Sin embargo, puede que la transacción no sea secuenciable con respecto a otras. Por ejemplo, cuando se buscan datos que cumplan ciertas condiciones, una transacción puede encontrar algunos datos insertados por una transacción comprometida, pero puede no encontrar otros datos que haya insertado la misma transacción.
- **Lectura comprometida:** solo permite leer datos comprometidos, pero no requiere lecturas repetibles. Por ejemplo, entre dos lecturas de un elemento de datos en una transacción, otra transacción podría actualizar el elemento de datos y comprometerlo.
- **Lectura no comprometida:** permite leer datos no comprometidos. Es el nivel de aislamiento más bajo que permite SQL.

Ninguno de los niveles de aislamiento anteriores permite **escrituras sucias,** es decir, no permite escribir en un elemento de datos que ya ha sido escrito por otra transacción que no se ha comprometido o abortado.

Muchos sistemas de bases de datos se ejecutan, de forma predeterminada, en el nivel de aislamiento lectura comprometida. En SQL se puede indicar explícitamente el nivel de aislamiento, en lugar de aceptar la configuración predeterminada del sistema. Por ejemplo, la sentencia «**set transaction isolation level serializable**» establece el nivel de aislamiento como secuenciable; se puede especificar cualquiera de los otros niveles. La sintaxis anterior está disponible en Oracle, PostgreSQL y SQL Server; DB2 usa la sintaxis «**change isolation level**» con sus propias abreviaturas para los niveles de aislamiento.

El cambio del nivel de aislamiento debe ser la primera sentencia de una transacción. Más aún, se debe desactivar el compromiso automático de las sentencias individuales, si está por defecto; las funciones de API, como el método de JDBC `Connection.setAutoCommit(false)` que se trató en la Sección 5.1.1.7, se puede usar para ello. Además, en JDBC se puede usar el método `Connection.setTransactionIsolation(int level)` para establecer el nivel de aislamiento; consulte los manuales de JDBC para más detalles.

El diseñador de la aplicación puede decidir aceptar un nivel de aislamiento más débil para mejorar el rendimiento del sistema. Como se verá en la Sección 14.9 y en el Capítulo 15, asegurar la secuencialidad puede forzar a que una transacción tenga que esperar por otra o, en algunos casos, tenga que abortar porque ya no se pueda ejecutar como parte de una ejecución secuenciable. Aunque pueda parece poco riguroso arriesgar la consistencia de la base de datos por el rendimiento, este compromiso puede tener sentido si se puede asegurar que la inconsistencia que se pueda producir no es relevante para la aplicación.

Existen muchos medios para implementar los niveles de aislamiento. Siempre que la implementación asegure la secuencialidad, el diseñador de la aplicación de la base de datos o el usuario de las aplicaciones no necesitan conocer los detalles de la implementación, salvo quizás para tratar los problemas de rendimiento. Desafortunadamente, aunque el nivel de aislamiento se haya establecido a **secuenciable,** algunas bases de datos implementan realmente un nivel de aislamiento más débil, que no rechaza todas las posibles ejecuciones no secuenciables; se volverá a tratar este tema en la Sección 14.9. Si se usan niveles de aislamiento más débiles, explícitamente o implícitamente, el diseñador de aplicaciones debe ser consciente de algunos detalles de implementación para evitar o minimizar la posibilidad de inconsistencias debidas a la falta de secuencialidad.

14.9. Implementación de niveles de aislamiento

Hasta este momento se han visto las propiedades que debe tener una planificación para dejar a la base de datos en un estado consistente y para permitir que los fallos en las transacciones se puedan manejar de una manera segura.

Existen varias directivas de **control de concurrencia** que se pueden utilizar para asegurar que, incluso si se ejecutan concurrentemente muchas transacciones, solo se generen planificaciones aceptables sin tener en cuenta la forma en que el sistema operativo comparte en el tiempo los recursos (tales como el tiempo de CPU) entre las transacciones.

Como ejemplo trivial de directiva de control de concurrencia considere el siguiente: una transacción obtiene un **bloqueo** de la base de datos completa antes de comenzar y lo libera después de haber comprometido. Mientras una transacción mantiene el bloqueo, no se permite que ninguna otra lo obtenga, y todas ellas deben esperar hasta que se libere el bloqueo. Como resultado de esta política de bloqueo, solo se puede ejecutar una transacción cada vez. Por tanto, solo se generan planificaciones secuenciales. Estas son obviamente secuenciables y es sencillo probar que también son sin cascada.

Un esquema de control de concurrencia como este produce un rendimiento pobre, ya que fuerza a que las transacciones esperen a que finalicen las precedentes para poder comenzar. En otras palabras, produce un grado de concurrencia pobre. Como se ha explicado en la Sección 14.5, la ejecución concurrente presenta ventajas en cuanto a rendimiento.

El objetivo de las directivas de control de concurrencia es proporcionar un elevado grado de concurrencia, al mismo tiempo que aseguran que todas las planificaciones que se generan son secuenciables en cuanto a conflictos o en cuanto a vistas y son sin cascada.

En este capítulo se ofrece una panorámica general de cómo funcionan los mecanismos de control de concurrencia más importantes y se presentan los detalles en el Capítulo 15.

SECUENCIALIDAD EN EL MUNDO REAL

Las planificaciones secuenciables son las ideales para garantizar la consistencia, pero en el día a día no se imponen requisitos tan rígidos. Un sitio web que ofrezca bienes en venta puede listar un elemento como en almacén, y en el tiempo que transcurre entre que el usuario selecciona el elemento y va al proceso de pago ese elemento puede que ya no esté disponible. Desde el punto de vista de la base de datos, sería una lectura no repetible.

Otro ejemplo, suponga una selección de asiento en un vuelo. Imagine que un viajero ya ha reservado el itinerario y ahora está seleccionando asientos para los vuelos. Muchos sitios web de aerolíneas permiten al usuario elegir distintos vuelos y elegir asientos, tras lo cual se pregunta al usuario que confirme la selección. Pudiera ocurrir que otro pasajero estuviese seleccionando o cambiando su selección de asiento para los mismos vuelos durante el mismo tiempo. La disponibilidad de asientos que se muestra al viajero estaba cambiando, pero al viajero se le muestra una vista fija de la disponibilidad como cuando empezó el proceso de selección.

Incluso aunque dos viajeros seleccionen asientos a la vez, lo más probable es que seleccionen asientos diferentes y, siendo así, no existe ningún conflicto real. Sin embargo, las transacciones no son secuenciables, ya que cada viajero ha leído datos que fueron posteriormente modificados por otro viajero, lo que puede conducir a un ciclo en el grafo de precedencia. Si dos viajeros que realizan la selección de asientos concurrentemente eligen realmente el mismo asiento, uno de ellos no podrá conseguir el asiento que había seleccionado; sin embargo, la situación se puede resolver fácilmente pidiendo al viajero que seleccione otro asiento, con la información de disponibilidad actualizada.

Es posible forzar la secuencialidad permitiendo que solo un viajero a la vez pueda realizar una selección de asiento para un determinado vuelo. Sin embargo, haciéndolo así podría ocasionar retrasos importantes en otros viajeros que tendrían que esperar a que su vuelo estuviese disponible para la selección de sus asientos; en particular, un viajero que tardase mucho tiempo en realizar la selección podría generar un problema serio para otros viajeros. En su lugar, este tipo de transacciones se dividen en una parte que requiere la interacción con el usuario y una parte que se ejecuta exclusivamente en la base de datos. En el ejemplo anterior, la transacción con la base de datos comprobaría los asientos elegidos para ver si siguen disponibles y, si es así, actualizarlos en la base de datos. La secuencialidad solo se asegura para aquellas transacciones que se ejecutan en la base de datos, sin interacción con el usuario.

14.9.1. Bloqueo

En lugar de bloquear toda la base de datos, una transacción podría bloquear solo aquellos elementos de datos a los que acceda. Con esta directiva, la transacción debe mantener bloqueos el tiempo necesario para asegurar la secuencialidad, pero durante un tiempo suficientemente corto como para que no dañe excesivamente el rendimiento. La complicación surge con sentencias de SQL como las que se vieron en la Sección 14.10, en las que los elementos a los que se accede dependen de una cláusula **where.** En el Capítulo 15 se presenta el protocolo de bloqueo en dos fases, una primera fase en la que adquiere bloqueos pero no libera ninguno y una segunda fase en la que la transacción libera los bloqueos pero no adquiere ninguno. (En la práctica, los bloqueos se suelen liberar solo cuando la transacción termina su ejecución y ya se ha comprometido o abortado).

Si se dispone de dos tipos distintos de bloqueos se pueden conseguir mejoras en el bloqueo: los compartidos y los exclusivos. Los bloqueos compartidos se usan para los datos que la transacción lee y los bloqueos exclusivos para aquellos que escribe. Varias transacciones pueden mantener bloqueos compartidos en el mismo elemento de datos a la vez, pero se concede un bloqueo exclusivo a una transacción sobre un elemento de datos solo si ninguna otra tiene un bloqueo (independientemente de si es exclusivo o compartido). Este uso de los dos modos de bloqueos junto con el bloqueo en dos fases permite la lectura concurrente de datos mientras aún se asegura la secuencialidad.

14.9.2. Marcas de tiempo

Otra categoría de técnicas para la implementación del aislamiento asigna a cada transacción una **marca de tiempo,** normalmente cuando comienza. Para cada elemento de datos, el sistema mantiene dos marcas de tiempo. La marca de tiempo de lectura guarda la marca de tiempo más grande (la más reciente) de las transacciones que leyeron el dato. La marca de tiempo de escritura del dato mantiene la marca de tiempo de la transacción que escribió el valor actual del dato. Las marcas de tiempo se usan para asegurar que las transacciones acceden a los datos en el orden de las marcas de tiempo de las transacciones si se produce un conflicto en el acceso. Cuando no es posible, las transacciones beligerantes se abortan y se reinician con una nueva marca de tiempo.

14.9.3. Versiones múltiples y aislamiento de instantáneas

Si se mantiene más de una versión de un dato, se puede permitir que una transacción lea la versión antigua del dato en lugar de la nueva que ha escrito una transacción comprometida o una transacción que debería no ser posterior en el orden de secuenciación. Existen diversas técnicas de control de concurrencia multiversión. En la práctica hay una ampliamente utilizada que se denomina **aislamiento de instantáneas.**

En el aislamiento de instantáneas, se puede imaginar que a cada transacción se le da su propia versión, o instantánea, de la base de datos cuando comienza.[4] Lee los datos de su propia versión privada y, por tanto, está aislada de las actualizaciones de otras transacciones. Si la transacción actualiza la base de datos, esa actualización solo aparece en su propia versión, no en la base de datos real. La información de las actualizaciones se guarda de forma que se puedan aplicar a la base de datos «real» si la transacción se compromete.

Cuando una transacción *T* entra en el estado de parcialmente comprometida, solo procede al estado de comprometida si y solo si ninguna otra transacción ha modificado el dato que *T* intentaba actualizar. Las transacciones que, como resultado, no consiguen comprometerse, abortan.

El aislamiento de instantáneas asegura que los intentos de leer los datos nunca tienen que esperar (al contrario que con el bloqueo). Las transacciones que solo realizan lecturas no se pueden abortar; solo las que modifican los datos corren el ligero riesgo de abortar. Como cada transacción lee su propia versión, o instantánea, de la base de datos, la lectura de datos no hace que los intentos siguientes de actualización de otras transacciones tengan que esperar (al contrario que con el bloqueo). Como la mayoría de las transacciones son de lectura (y el resto suele leer más datos que los que actualizan), suele ser un mecanismo de mejora significativa del rendimiento, comparado con los bloqueos.

El problema con el aislamiento de instantáneas es que, paradójicamente, proporciona *demasiado* aislamiento. Suponga dos transacciones *T* y *T'*. En una ejecución secuenciable, o bien *T* ve todas las actualizaciones que hace *T'*, o bien *T'* ve todas las actualizaciones que hace *T*, ya que una debe seguir a la otra en el orden de secuenciación. Con el aislamiento de instantáneas, se dan casos en los que ninguna de las transacciones ve las actualizaciones de la otra. Es una situación que no puede ocurrir en una ejecución secuenciable. En muchos casos (en realidad, en la mayoría) los accesos a los datos por las dos transacciones no entran en conflicto y, por tanto, no hay problemas. Sin embargo, si *T* lee datos que *T'* actualiza y *T'* lee datos que *T* actualiza, es posible que ambas transacciones fallen en la lectura de las actualizaciones que hizo la otra. El resultado, como se verá en el Capítulo 15, puede crear un estado inconsistente de la base de datos que, por supuesto, no se puede conseguir con ninguna ejecución secuenciable.

Oracle, PostgreSQL y SQL Server ofrecen la opción de aislamiento de instantáneas. Oracle y PostgreSQL implementan el nivel de aislamiento **secuenciable** usando el aislamiento de instantáneas. Como resultado, su implementación de la secuencialidad puede, en circunstancias excepcionales, generar que se permita una ejecución no secuenciable. Sin embargo, SQL Server incluye un nivel de aislamiento adicional que va más allá de los niveles estándar, llamado **instantánea** *(snapshot)*, para ofrecer la opción de aislamiento de instantáneas.

14.10. Transacciones como sentencias de SQL

En la Sección 4.3 se presentó la sintaxis de SQL para especificar el comienzo y el final de una transacción. Ahora que ya se han visto los elementos que permiten asegurar las propiedades ACID para transacciones, se va a ver cómo asegurar que se cumplen estas propiedades cuando se especifica una transacción como una secuencia de sentencias de SQL en lugar del modelo tan restrictivo de lecturas y escrituras que hemos considerado hasta el momento.

4 En la realidad, por supuesto, no se copia la base de datos completa. Solo se mantienen múltiples versiones de los elementos de datos que cambian.

En este modelo simple, se ha supuesto que existen un conjunto de datos. Aunque en el modelo se permite modificar los valores de los datos, no se permite que se puedan crear ni eliminar. Sin embargo, en SQL la sentencia **insert** crea nuevos datos y la sentencia **delete** los elimina. De hecho, estas dos sentencias son operaciones de **escritura,** ya que modifican la base de datos, pero sus interacciones con las acciones de otras transacciones son diferentes de las que se han visto en el modelo simple anterior. A modo de ejemplo, considere la siguiente sentencia de SQL sobre la base de datos de la universidad que busca todos los profesores cuyo sueldo sea superior a 90.000 €.

select *ID*, *nombre*
from *profesor*
where *sueldo* > 90000;

Utilizando la relación *profesor* del ejemplo (véase Apéndice A.3, Fig. A.16), se puede encontrar que solo Einstein y Brandt cumplen la condición. Suponga ahora que en ese mismo momento que se está ejecutando esta consulta otro usuario inserta a un nuevo profesor de nombre «James» con un sueldo de 100.000 €.

insert into *profesor* **values** ('11111', 'James', 'Marketing', 100000);

El resultado de la consulta puede ser diferente dependiendo de si la inserción se realiza antes o después de que se ejecute nuestra búsqueda. En una ejecución concurrente de estas dos transacciones, está claro que entran en conflicto, pero es un tipo de conflicto que no queda recogido en nuestro modelo simple. Esta situación se define como **fenómeno fantasma,** ya que el conflicto existe en datos «fantasma».

Nuestro modelo simple de transacciones requería que las operaciones se realizasen sobre datos concretos que se indicaban como argumentos de la operación. En dicho modelo, se pueden observar los pasos de **lectura** y **escritura** para ver a qué datos se hace referencia. Pero en una sentencia de SQL, los datos concretos (las tuplas) a los que se hace referencia pueden venir determinados por un predicado con cláusula **where.** Por tanto, una misma transacción, que se ejecuta más de una vez, podría hacer referencia a distintos datos cada vez si los valores de la base de datos cambian entre ejecuciones.

Una forma de tratar con el problema anterior es darse cuenta de que para el control de concurrencia no es suficiente considerar solamente las tuplas a las que accede una transacción; también hay que considerar la información que se usa para encontrar las tuplas a las que acceder. La información que se usa para encontrar las tuplas puede cambiar por una inserción o un borrado, o en el caso de un índice, incluso por una actualización de un atributo de la clave de búsqueda. Por ejemplo, si se utiliza un bloqueo para el control de concurrencia, también hay que bloquear las estructuras de datos que realizan el seguimiento de las tuplas de una relación, así como las estructuras de los índices. Sin embargo, este tipo de bloqueo puede conducir a una degradación de la concurrencia en algunas situaciones; en la Sección 15.8.3 se ven los protocolos de bloqueo de índices que maximizan la concurrencia, asegurando la secuencialidad aun cuando se produzcan inserciones, borrados o predicados en las consultas.

Suponga de nuevo la siguiente consulta:

select *ID*, *nombre*
from *profesor*
where *sueldo* > 90000;

y la siguiente actualización en SQL:

update *profesor*
set *sueldo* = *sueldo* * 0.9
where *nombre* = 'Wu';

Nos enfrentamos a una interesante situación para determinar si nuestra consulta entra en conflicto con la sentencia de actualización. Si la consulta lee la relación *profesor* completa, entonces lee también la tupla con los datos de Wu y entra en conflicto con la actualización. Sin embargo, si existe un índice que permita acceder directamente a las tuplas con *sueldo* > 90.000 €, entonces la consulta no habría accedido a los datos de Wu, ya que el sueldo inicial de Wu es de 90.000 € en el ejemplo de la relación profesor, y la actualización reduce su sueldo a 81.000 €.

Sin embargo, usando el enfoque anterior, parecería que la existencia del conflicto depende de una decisión de bajo nivel del procesamiento de la consulta que no está relacionado con el nivel de usuario sobre el significado de las sentencias de SQL. Un enfoque alternativo al control de la concurrencia trata las inserciones, borrados o actualizaciones como conflictivas con un predicado sobre una relación si pueden afectar al conjunto de tuplas seleccionadas por un predicado. En el ejemplo anterior, el predicado es «*sueldo* > 90000», y una actualización del sueldo de Wu desde 90.000 € a un valor superior a 90.000 €, o una actualización del sueldo de Einstein desde un valor superior a 90.000 € a un valor menor o igual a 90.000 €, podría entrar en conflicto con este predicado. El bloqueo que se basa en esta idea se denomina **bloqueo de predicado;** sin embargo, el bloqueo de predicado es muy costoso y en la práctica no se utiliza.

14.11. Resumen

- Una *transacción* es una *unidad* de la ejecución de un programa que accede y posiblemente actualiza varios elementos de datos. Es fundamental comprender el concepto de transacción para entender e implementar las actualizaciones de los datos en una base de datos, de manera que las ejecuciones concurrentes y los fallos de varios tipos no den como resultado que la base de datos se vuelva inconsistente.
- Es necesario que las transacciones tengan las propiedades ACID: atomicidad, consistencia, aislamiento y durabilidad.
  - La atomicidad asegura que, o bien todos los efectos de la transacción se reflejan en la base de datos, o bien ninguno de ellos lo hace; un fallo no puede dejar a la base de datos en un estado en el cual una transacción se haya ejecutado parcialmente.
  - La consistencia asegura que si la base de datos es consistente inicialmente, la ejecución de la transacción (debido a la misma) deja a la base de datos en un estado consistente.
  - El aislamiento asegura que en la ejecución concurrente de transacciones, estas están aisladas entre sí, de tal manera que cada una tiene la impresión de que ninguna otra transacción se ejecuta concurrentemente con ella.
  - La durabilidad asegura que, una vez que la transacción se ha comprometido, las actualizaciones hechas por la transacción no se pierden incluso aunque se produzca un fallo del sistema.
- La ejecución concurrente de transacciones mejora el rendimiento y la utilización del sistema, y también reduce el tiempo de espera de las transacciones.
- Los distintos tipos de almacenamiento en una computadora son el almacenamiento volátil, el no volátil y el estable. Los datos en almacenamiento volátil, como la RAM, se pierden cuando la computadora falla. Los datos en almacenamiento no volátil, como los discos, no se pierden cuando la computadora falla, pero pueden perderse ocasionalmente debido a fallos en los discos. Los datos en almacenamiento estable nunca se pierden.
- El almacenamiento estable que se debe hacer accesible en línea se puede realizar con discos espejo u otras formas de RAID, que proporcionan almacenaje de datos redundantes. El almacenamiento estable fuera de línea, o archivado, puede constar de varias copias en cintas de los datos almacenados en una localización físicamente segura.
- Cuando varias transacciones se ejecutan concurrentemente en la base de datos, puede que deje de conservarse la consistencia de los datos. Es por tanto necesario que el sistema controle la interacción entre las transacciones concurrentes.
  - Puesto que una transacción es una unidad que conserva la consistencia, una ejecución secuencial de transacciones garantiza que se conserve dicha consistencia.
  - Una *planificación* captura las acciones clave de las transacciones que afectan a la ejecución concurrente, tales como las operaciones leer y escribir, a la vez que se abstraen los detalles internos de la ejecución de la transacción.
  - Es necesario que toda planificación producida por el procesamiento concurrente de un conjunto de transacciones tenga el efecto equivalente a una planificación en la cual esas transacciones se ejecutan secuencialmente en un cierto orden.
  - Un sistema que asegure esta propiedad se dice que asegura la *secuencialidad*.
  - Existen varias nociones distintas de equivalencia que llevan a los conceptos de *secuencialidad en cuanto a conflictos* y *secuencialidad en cuanto a vistas*.
- Se puede asegurar la secuencialidad de las planificaciones generadas por la ejecución concurrente de transacciones mediante una gran variedad de mecanismos llamados directivas de *control de concurrencia*.
- Se puede comprobar si una planificación es secuenciable en cuanto a conflictos construyendo el *grafo de precedencia* para dicha planificación y viendo que no hay ciclos en el grafo. Sin embargo, hay esquemas de control de concurrencia más eficientes para asegurar la secuencialidad.
- Las planificaciones deben ser recuperables para asegurar que si la transacción a observa los efectos de la transacción b y esta aborta, entonces a también se aborte.
- Las planificaciones deben ser preferentemente sin cascada para que el hecho de abortar una transacción no provoque abortos en cascada de otras transacciones. Para asegurar que sean sin cascada, las transacciones solo deben leer datos comprometidos.
- El componente de gestión de control de concurrencia de la base de datos es el responsable de manejar las directivas de control de concurrencia. En el Capítulo 15 se describen las directivas de control de concurrencia.

Términos de repaso

- Transacción.
- Propiedades ACID.
  - Atomicidad.
  - Consistencia.
  - Aislamiento.
  - Durabilidad.
- Estado inconsistente.
- Tipos de almacenamiento.
  - Almacenamiento volátil.
  - Almacenamiento no volátil.
  - Almacenamiento estable .
- Sistema de control de la concurrencia.
- Sistema de recuperación.
- Estados de una transacción.
  - Activa.
  - Parcialmente comprometida.
  - Fallida.
  - Abortada.
  - Comprometida.
  - Terminada.
- Transacción.
  - Reiniciar.
  - Cancelar.
- Escrituras externas observables.
- Ejecuciones concurrentes.
- Ejecución secuencial.
- Planificaciones.
- Conflictos entre operaciones.
- Equivalencia en cuanto a conflictos.
- Secuencialidad en cuanto a conflictos.
- Determinación de la secuencialidad.
- Grafo de precedencia.
- Orden de secuencialidad.
- Planificaciones recuperables.
- Retroceso en cascada.
- Planificaciones sin cascada.
- Esquema de control de concurrencia.
- Bloqueo.
- Versiones múltiples.
- Aislamiento de instantáneas.

Ejercicios prácticos

14.1. Suponga que existe un sistema de base de datos que nunca falla. ¿Se necesita un gestor de recuperaciones para este sistema?

14.2. Considere un sistema de archivos como el de su sistema operativo preferido.

a. ¿Cuáles son los pasos involucrados en la creación y borrado de archivos y en la escritura de datos a archivos?

b. Explique por qué son relevantes los aspectos de atomicidad y durabilidad en la creación y borrado de archivos y en la escritura de datos a archivos.

14.3. Los implementadores de sistemas de bases de datos prestan mucha más atención a las propiedades ACID que los implementadores de sistemas de archivos. ¿Por qué tiene sentido esto?

14.4. Justifique lo siguiente: la ejecución concurrente de transacciones es más importante cuando los datos se deben extraer del disco (lento) o cuando las transacciones duran mucho, y es menos importante cuando hay pocos datos en la memoria y las transacciones son muy cortas.

14.5. Puesto que toda planificación secuenciable en cuanto a conflictos es secuenciable en cuanto a vistas, ¿por qué se hace hincapié en la secuencialidad en cuanto a conflictos en vez de en la secuencialidad en cuanto a vistas?

14.6. Considere el grafo de precedencia de la Figura 14.16. ¿Es secuenciable en cuanto a conflictos la planificación correspondiente? Razone la respuesta.

14.7. ¿Qué es una planificación sin cascada? ¿Por qué es conveniente la planificación sin cascada? ¿Hay circunstancias bajo las cuales puede ser conveniente permitir planificaciones que no sean sin cascada? Razone la respuesta.

14.8. La anomalía de **actualización perdida** se dice que se produce si una transacción T_j lee un dato, entonces otra transacción T_k escribe el dato (posiblemente debido a una lectura previa), tras lo cual T_j escribe el dato. La actualización que ha realizado T_k se ha perdido, ya que la actualización que ha hecho T_j ignora el valor escrito por T_k.

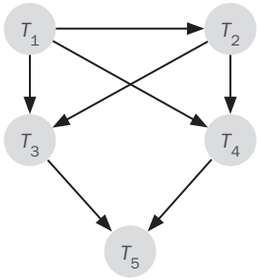


Figura 14.16. Grafo de precedencia para el Ejercicio práctico 14.6.

a. Indique una planificación de ejemplo que muestre esta anomalía.

b. Muestre en la planificación de ejemplo como es posible que se produzca la anomalía de actualización perdida con el nivel de aislamiento de **lectura comprometida.**

c. Explique por qué no se puede producir la anomalía de actualización perdida con el nivel de aislamiento de **lecturas repetibles.**

14.9. Considere una base de datos para un banco en el que el sistema de bases de datos usa un aislamiento de instantánea. Describa un escenario concreto en el que se puede producir una ejecución no secuenciable que puede generar un problema para el banco.

14.10. Considere una base de datos para una línea aérea en la que el sistema de bases de datos usa un aislamiento de instantánea. Describa un escenario concreto en el que se puede producir una ejecución no secuenciable, pero la línea aérea podría aceptarla para poder conseguir un mejor rendimiento global.

14.11. La definición de una planificación supone que las operaciones están totalmente ordenadas en el tiempo. Considere un sistema de bases de datos con múltiples procesadores en el que no es posible establecer siempre una ordenación exacta entre las operaciones que se ejecutan entre los distintos procesadores. Sin embargo, las operaciones sobre un elemento de datos pueden estar totalmente ordenadas.

¿Genera la situación indicada algún problema con la definición de secuencialidad en cuanto a conflictos? Justifique su respuesta.

Ejercicios

14.12. Enumere las propiedades ACID. Explique la utilidad de cada una.

14.13. Durante su ejecución, una transacción pasa a través de varios estados hasta que se compromete o aborta. Indique todas las secuencias posibles de estados por los que puede pasar una transacción. Explique por qué puede producirse cada una de las transiciones de estados.

14.14. Explique la diferencia entre los términos *planificación secuencial* y *planificación secuenciable*.

14.15. Considere las dos transacciones siguientes:

T_{13} : leer(A);
leer(B);
if $A = 0$ **then** $B := B + 1$;
escribir(B)

T_{14} : leer(B);
leer(A);
if $B = 0$ **then** $A := A + 1$;
escribir(A)

Sea el requisito de consistencia $A = 0 \vee B = 0$, siendo los valores iniciales $A = B = 0$.

a. Demuestre que toda ejecución secuencial en la que aparezcan estas transacciones conserva la consistencia de la base de datos.

b. Muestre una ejecución concurrente de T_{13} y T_{14} que produzca una planificación no secuenciable.

c. ¿Existe una ejecución concurrente de T_{13} y T_{14} que produzca una planificación secuenciable?

14.16. Indique un ejemplo de una planificación secuenciable con dos transacciones tales que el orden en que se comprometen sea diferente del orden de secuenciación.

14.17. ¿Qué es una planificación recuperable? ¿Por qué es conveniente la recuperabilidad de las planificaciones? ¿Hay circunstancias bajo las cuales puede ser conveniente permitir planificaciones no recuperables? Razone la respuesta.

14.18. ¿Por qué los sistemas de bases de datos permiten la ejecución concurrente de transacciones, a pesar del esfuerzo de programación necesario para asegurar que la ejecución concurrente no causa ningún problema?

14.19. Explique por qué el nivel de aislamiento de lectura comprometida asegura que las planificaciones son sin cascada.

14.20. Para cada uno de los siguientes niveles de aislamiento, indique un ejemplo de una planificación que respete los niveles de aislamiento indicados, pero no sea secuenciable.

a. Lectura no comprometida.

b. Lectura comprometida.

c. Lectura repetible.

14.21. Suponga que además de las operaciones leer y escribir se permite la operación pred_leer(r, P), que lee todas las tuplas de la relación r que satisfacen el predicado P.

a. Indique un ejemplo de una planificación que use la operación pred_leer que presente el fenómeno fantasma y, como consecuencia, resulte no secuenciable.

b. Indique un ejemplo de una planificación en la que una transacción use la operación pred_leer sobre la relación r y otra transacción borre una tupla de r, pero la planificación no presente el conflicto fantasma. (Para ello debe indicar un esquema de relación r, y mostrar los valores de los atributos de las tuplas borradas).

Notas bibliográficas

Gray y Reuter [1993] proporcionan un extenso tratamiento de los conceptos y técnicas de procesamiento de transacciones, las técnicas y detalles de la implementación, incluyendo resultados de la recuperación y el control de concurrencia. Bernstein y Newcomer [1997] proporcionan varios aspectos del procesamiento de transacciones.

El concepto de secuencialidad fue formalizado por Eswaran et ál. [1976] en conexión con su trabajo sobre el control de concurrencia para System R.

Otras referencias que cubren aspectos específicos del procesamiento de transacciones, tales como el control de concurrencia y la recuperación, se citan en los Capítulos 15, 16 y 26.

Control de concurrencia 15

En el Capítulo 14 se estudió que una de las propiedades fundamentales de las transacciones es el aislamiento. Cuando se ejecutan varias transacciones concurrentemente en la base de datos, puede que deje de conservarse la propiedad de aislamiento. Es necesario que el sistema controle la interacción entre las transacciones concurrentes; dicho control se lleva a cabo a través de uno de los muchos mecanismos existentes llamado esquema de *control de concurrencia*. En el Capítulo 26 se tratan los esquemas de control de concurrencia que admiten planificaciones no secuenciables e ignoran los fallos. En el Capítulo 16 se puede ver cómo el sistema se puede recuperar de los fallos.

Como se verá, existen distintos esquemas de control de concurrencia. Ninguno es claramente el mejor; cada uno tiene sus ventajas. En la práctica, los esquemas que más se utilizan son el *bloqueo en dos fases* y el *aislamiento de instantáneas*.

15.1. Protocolos basados en el bloqueo

Una forma de asegurar la secuencialidad es exigir que el acceso a los elementos de datos se haga en exclusión mutua; es decir, mientras una transacción accede a un elemento de datos, ninguna otra transacción puede modificar dicho elemento. El método más habitual que se usa para implementar este requisito es permitir que una transacción acceda a un elemento de datos solo si posee actualmente un **bloqueo** sobre dicho elemento. En la Sección 14.9 se introdujo el concepto de bloqueo.

15.1.1. Bloqueos

Existen varios modos mediante los cuales se puede bloquear un elemento de datos. En esta sección se centra la atención en dos de dichos modos:

1. **Compartido.** Si una transacción T_i obtiene un **bloqueo en modo compartido** (denotado por C) sobre el elemento Q, entonces T_i puede leer Q, pero no lo puede escribir.
2. **Exclusivo.** Si una transacción T_i obtiene un **bloqueo en modo exclusivo** (denotado por X) sobre el elemento Q, entonces T_i puede tanto leer como escribir Q.

Es necesario que toda transacción **solicite** un bloqueo del modo apropiado sobre el elemento de datos Q, dependiendo de los tipos de operaciones que se vayan a realizar sobre Q. La petición se hace al gestor de control de concurrencia. La transacción puede realizar la operación solo después de que el gestor de control de concurrencia **conceda** el bloqueo a la transacción. El uso de estos dos modos de bloqueo permite que varias transacciones puedan leer un elemento de dato pero limita el acceso de escritura a solo una transacción a la vez.

Para indicarlo de forma más general, dado un conjunto de modos de bloqueo, se puede definir sobre ellos una **función de compatibilidad,** como se explica a continuación. Sean A y B para representar dos modos de bloqueo arbitrarios. Suponga que la transacción T_i solicita un bloqueo en modo A sobre el elemento Q bajo el que la transacción T_j ($T_i \neq T_j$) posee actualmente un bloqueo de modo B. Si a la transacción T_i se le puede conceder un bloqueo sobre Q inmediatamente, a pesar de la presencia del bloqueo de modo B, entonces se dice que el modo A es **compatible** con el modo B. Tal función se puede representar convenientemente en forma de matriz. La relación de compatibilidad entre los dos modos de bloqueo que se usan en esta sección se presenta en la matriz comp de la Figura 15.1. Un elemento comp(A, B) de la matriz tiene el valor *cierto* si, y solo si, el modo A es compatible con el modo B.

	C	X
C	cierto	falso
X	falso	falso

Figura 15.1. Matriz de compatibilidad de bloqueos en modo C.

Obsérvese que el modo compartido es compatible consigo mismo, pero no con el modo exclusivo. En todo momento se pueden tener varios bloqueos en modo compartido (por varias transacciones) sobre un elemento de datos en concreto. Una petición posterior de bloqueo en modo exclusivo debe esperar hasta que los bloqueos actuales en modo compartido sean liberados.

Una transacción solicita un bloqueo compartido sobre el elemento de datos Q a través de la instrucción bloquear-c(Q). De forma similar se solicita un bloqueo exclusivo a través de la instrucción bloquear-x(Q). Se puede desbloquear un elemento de datos Q por medio de la instrucción desbloquear(Q).

Para acceder a un elemento de datos, una transacción T_i debe en primer lugar bloquear dicho elemento. Si este ya se encuentra bloqueado por otra transacción en un modo incompatible, el gestor de control de concurrencia no concederá el bloqueo hasta que se hayan liberado todos los bloqueos incompatibles que posean otras transacciones. De este modo, T_i debe **esperar** hasta que se liberen todos los bloqueos incompatibles que posean otras transacciones.

La transacción T_i puede desbloquear un elemento de datos que haya bloqueado en algún momento anterior. Observe que la transacción debe mantener un bloqueo sobre un elemento de datos mientras acceda a dicho elemento. Además, no siempre es aconsejable que una transacción desbloquee un elemento de datos inmediatamente después de finalizar su acceso sobre él, ya que puede dejar de asegurarse la secuencialidad.

```
T1: bloquear-x(B);
    leer(B);
    B := B − 50;
    escribir(B);
    desbloquear(B);
    bloquear-x(A);
    leer(A);
    A := A + 50;
    escribir(A);
    desbloquear(A).
```

Figura 15.2. Transacción T_1.

Como ejemplo, considere de nuevo el sistema bancario simplificado que se presentó en el Capítulo 14. Sean A y B dos cuentas a las que acceden las transacciones T_1 y T_2. La transacción T_1 transfiere 50 € desde la cuenta B a la A (Figura 15.2). La transacción T_2 visualiza la cantidad total de dinero de las cuentas A y B; es decir, la suma $A + B$ (Figura 15.3).

Suponga que los valores de las cuentas A y B son 100 € y 200 €, respectivamente. Si estas dos transacciones se ejecutan secuencialmente, tanto en el orden T_1 T_2 como en el orden T_2 T_1, entonces la transacción T_2 visualizará el valor 300 €. Si por el contrario estas transacciones se ejecutan concurrentemente, entonces puede darse la planificación 1, que se muestra en la Figura 15.4. En ese caso la transacción T_2 visualiza 250 €, lo cual es incorrecto. El motivo por el que se produce esta incorrección es que la transacción T_1 desbloquea el elemento B demasiado pronto, lo cual provoca que T_2 perciba un estado inconsistente.

La planificación muestra las acciones que ejecuta cada transacción, así como los puntos en los que el gestor de control de concurrencia concede los bloqueos. La transacción que realiza una petición de bloqueo no puede ejecutar su siguiente acción hasta que el gestor de control de concurrencia conceda dicho bloqueo. Por tanto, el bloqueo debe concederse en el intervalo de tiempo entre la operación de petición del bloqueo y la siguiente acción de la transacción. No es importante el momento exacto del intervalo en el cual se produce la concesión del bloqueo; se puede asumir sin problemas que la concesión del bloqueo se produce justo antes de la siguiente acción de la transacción. Por tanto, se va a obviar la columna que describe las acciones del gestor de control de concurrencia en todas las planificaciones que aparezcan en el resto del capítulo. Se deja al lector como ejercicio la deducción del momento en que se conceden los bloqueos.

```
T2: bloquear-c(A);
    leer(A);
    desbloquear(A);
    bloquear-c(B);
    leer(B);
    desbloquear(B);
    visualizar(A + B).
```

Figura 15.3. Transacción T_2.

T_1	T_2	Gestor de control de concurrencia
Bloquear-x(*B*)		
		conceder-x(*B*, T_1)
leer(*B*)		
B := *B* − 50		
escribir(*B*)		
desbloquear(*B*)		
	bloquear-c(*A*)	
		conceder-c(*A*, T_2)
	leer(*A*)	
	desbloquear(*A*)	
	bloquear-c(*B*)	
		conceder-c(*B*, T_2)
	leer(*B*)	
	desbloquear(*B*)	
	visualizar(*A* + *B*)	
bloquear-x(*A*)		
		conceder-x(*A*, T_1)
leer(*A*)		
A := *A* − 50		
escribir(*A*)		
desbloquear(*A*)		

Figura 15.4. Planificación 1.

Suponga ahora que el desbloqueo se retrasa hasta el final de la transacción. La transacción T_3 corresponde a T_1 con el desbloqueo retrasado (Figura 15.5). La transacción T_4 corresponde a T_2 con el desbloqueo retrasado (Figura 15.6).

Se puede verificar que la secuencia de lecturas y escrituras de la planificación 1, que provoca que se visualice un total incorrecto de 250 €, ya no es posible con T_3 y T_4. Son posibles otras planificaciones. T_4 no visualiza un resultado inconsistente en ninguna de ellas; más adelante se verá por qué.

```
T3: bloquear-x(B);
    leer(B);
    B := B − 50;
    escribir(B);
    bloquear-x(A);
    leer(A);
    A := A + 50;
    escribir(A);
    desbloquear(B);
    desbloquear(A).
```

Figura 15.5. Transacción T_3 (transacción T_1 con desbloqueo retrasado).

```
T4: bloquear-c(A);
    leer(A);
    bloquear-c(B);
    leer(B);
    visualizar(A + B);
    desbloquear(A);
    desbloquear(B).
```

Figura 15.6. Transacción T_4 (transacción T_2 con desbloqueo retrasado).

Desafortunadamente, el uso de bloqueos puede conducir a una situación no deseada. Considere la planificación parcial de la Figura 15.7 para T_3 y T_4. Puesto que T_3 posee un bloqueo sobre B en modo exclusivo y T_4 solicita un bloqueo sobre B en modo compartido, T_4 espera a que T_3 desbloquee B. De forma similar, puesto que T_4 posee un bloqueo sobre A en modo compartido y T_3 solicita un bloqueo sobre A en modo exclusivo, T_3 espera a que T_4 desbloquee A. Así se llega a un estado en el cual ninguna de las transacciones puede continuar su ejecución normal. Esta situación se denomina **interbloqueo.** Cuando aparece un interbloqueo, el sistema debe retroceder una de las dos transacciones. Una vez que se ha provocado el retroceso de una de ellas, se desbloquean los elementos de datos que estuvieran bloqueados por la transacción. Estos elementos de datos están disponibles entonces para otra transacción, la cual puede continuar su ejecución. Se volverá al tratamiento de interbloqueos en la Sección 15.2.

Si no se usan bloqueos, o se desbloquean los elementos de datos tan pronto como sea posible después de leerlos o escribirlos, se pueden obtener estados inconsistentes. Por otro lado, si no se desbloquea un elemento de datos antes de solicitar un bloqueo sobre otro, pueden producirse interbloqueos. Existen formas de evitar los interbloqueos en algunas situaciones, como se verá en la Sección 15.1.5. Sin embargo, en general, los interbloqueos son un mal necesario asociado a los bloqueos si se quieren evitar los estados inconsistentes. Los interbloqueos son absolutamente preferibles a los estados inconsistentes, ya que se pueden tratar haciendo retroceder las transacciones, mientras que los estados inconsistentes producen problemas en el mundo real que el sistema de base de datos no puede manejar.

T_3	T_4
bloquear-x*(B)*	
leer*(B)*	
$B := B - 50$	
escribir*(B)*	
	bloquear-c*(A)*
	leer*(A)*
	bloquear-c*(B)*
bloquear-x*(A)*	

Figura 15.7. Planificación 2.

Se exige que toda transacción del sistema siga un conjunto de reglas llamado **protocolo de bloqueo,** que indica el momento en que una transacción puede bloquear y desbloquear cada uno de los elementos de datos. Los protocolos de bloqueo restringen el número de planificaciones posibles. El conjunto de tales planificaciones es un subconjunto propio de todas las planificaciones secuenciables posibles. Se van a mostrar varios protocolos de bloqueo que solo permiten planificaciones secuenciables en cuanto a conflictos y que, por tanto, garantizan aislamiento. Antes de hacerlo se necesita cierta terminología.

Sean $\{T_0, T_1 \ldots , T_n\}$ un conjunto de transacciones que participan en la planificación S. Se dice que T_i **precede** a T_j en S, escrito como $T_i \rightarrow T_j$, si existe un elemento de datos Q tal que T_i ha obtenido un bloqueo en modo A sobre Q, y T_j ha obtenido un bloqueo en modo B sobre Q más tarde y comp(A, B) = falso. Si $T_i \rightarrow T_j$, entonces esta precedencia implica que en cualquier planificación secuencial equivalente, T_i debe aparecer antes que T_j. Observe que este grafo es similar al grafo de precedencia que se usaba en la Sección 14.6 para comprobar la secuencialidad en cuanto a conflictos. Los conflictos entre instrucciones corresponden a modos de bloqueo no compatibles.

Se dice que una planificación S es **legal** bajo un protocolo de bloqueo dado si S es una planificación posible para un conjunto de transacciones que sigan las reglas del protocolo de bloqueo. Se dice que un protocolo de bloqueo **asegura** la secuencialidad en cuanto a conflictos si, y solo si, todas las planificaciones legales son secuenciables en cuanto a conflictos; en otras palabras, para todas las planificaciones legales la relación → asociada es acíclica.

15.1.2. Concesión de bloqueos

Cuando una transacción solicita un bloqueo en un modo particular sobre un elemento de datos y ninguna otra transacción posee un bloqueo sobre el mismo elemento de datos en un modo conflictivo, se puede conceder el bloqueo. Sin embargo, hay que tener cuidado para evitar la siguiente situación: suponga que la transacción T_2 posee un bloqueo en modo compartido sobre un elemento de datos y que la transacción T_1 solicita un bloqueo en modo exclusivo sobre dicho elemento de datos. Obviamente, T_1 debe esperar a que T_2 libere el bloqueo en modo compartido. Mientras tanto, la transacción T_3 puede solicitar un bloqueo en modo compartido sobre el mismo elemento de datos. La petición de bloqueo es compatible con el bloqueo que se ha concedido a T_2, por tanto se puede conceder a T_3 el bloqueo en modo compartido. En este punto, T_2 puede liberar el bloqueo, pero T_1 debe seguir esperando hasta que T_3 termine. Pero de nuevo puede haber una nueva transacción T_4 que solicite un bloqueo en modo compartido sobre el mismo elemento de datos y a la cual se conceda el bloqueo antes de que T_3 lo libere. De hecho, es posible que haya una secuencia de transacciones que soliciten un bloqueo en modo compartido sobre el elemento de datos, y que cada una de ellas libere el bloqueo un poco después de que sea concedido, de forma que T_1 nunca obtenga el bloqueo en modo exclusivo sobre el elemento de datos. La transacción T_1 nunca progresará, y se dice que tiene **inanición.**

Se puede evitar la inanición de las transacciones concediendo los bloqueos de la siguiente manera: cuando una transacción T_i solicite un bloqueo sobre un elemento de datos Q en un modo particular M, el gestor de control de concurrencia concederá el bloqueo siempre que:

1. No exista otra transacción que posea un bloqueo sobre Q en un modo que entre en conflicto con M.
2. No exista otra transacción que esté esperando un bloqueo sobre Q y que lo haya solicitado antes que T_i.

De este modo, una petición de bloqueo nunca se quedará bloqueada por otra petición de bloqueo solicitada más tarde.

15.1.3. Protocolo de bloqueo de dos fases

Un protocolo que asegura la secuencialidad es el **protocolo de bloqueo de dos fases.** Este protocolo exige que cada transacción realice las peticiones de bloqueo y desbloqueo de dos fases:

1. **Fase de crecimiento.** Una transacción puede obtener bloqueos, pero no puede liberarlos.
2. **Fase de decrecimiento.** Una transacción puede liberar bloqueos, pero no puede obtener ninguno nuevo.

Inicialmente una transacción está en la fase de crecimiento. La transacción adquiere los bloqueos que necesite. Una vez que la transacción libera un bloqueo, entra en la fase de decrecimiento y no puede realizar más peticiones de bloqueo.

Por ejemplo, las transacciones T_3 y T_4 son de dos fases. Por otro lado, las transacciones T_1 y T_2 no son de dos fases. Observe que no es necesario que las instrucciones de desbloqueo aparezcan al final de la transacción. Por ejemplo, en el caso de la transacción T_3, se puede trasladar la instrucción desbloquear*(B)* hasta justo antes de la instrucción bloquear-x*(A)* y se sigue cumpliendo la propiedad del bloqueo de dos fases.

Se puede demostrar que el protocolo de bloqueo de dos fases asegura la secuencialidad en cuanto a conflictos. Considere cualquier transacción. El punto de la planificación en el cual la transacción obtiene su bloqueo final (el final de la fase de crecimiento) se denomina **punto de bloqueo** de la transacción. Ahora se pueden ordenar las transacciones según sus puntos de bloqueo; de hecho, esta ordenación es un orden de secuencialidad para las transacciones. La demostración se deja como ejercicio para el lector (véase el Ejercicio práctico 15.1).

El protocolo de bloqueo de dos fases *no* asegura la ausencia de interbloqueos. Observe que las transacciones T_3 y T_4 son de dos fases, pero en la planificación 2 (Figura 15.7) llegan a un interbloqueo.

Recuerde de la Sección 14.7.2 que las planificaciones, además de ser secuenciables, deberían ser sin cascada. El retroceso en cascada puede conducir al bloqueo de dos fases. Como ejemplo, considere la planificación parcial de la Figura 15.8. Cada transacción sigue el protocolo de bloqueo de dos fases, pero un fallo de T_5 después del paso leer(A) de T_7 lleva a un retroceso en cascada de T_6 y T_7.

Los retrocesos en cascada se pueden evitar mediante una modificación del protocolo de bloqueo de dos fases que se denomina **protocolo de bloqueo estricto de dos fases.** Este protocolo exige que, además de que el bloqueo sea de dos fases, una transacción deba poseer todos los bloqueos en modo exclusivo que tome hasta que dicha transacción se complete. Este requisito asegura que todo dato que escribe una transacción no comprometida está bloqueado en modo exclusivo hasta que la transacción se completa, evitando que ninguna otra transacción lea el dato.

Otra variante del bloqueo de dos fases es el **protocolo de bloqueo riguroso de dos fases,** el cual exige que se posean todos los bloqueos hasta que se comprometa la transacción. Se puede comprobar fácilmente que con el bloqueo riguroso de dos fases se pueden secuenciar las transacciones en el orden en que se comprometen.

T_5	T_6	T_7
bloquear-x*(A)*		
leer*(A)*		
bloquear-c*(B)*		
leer*(B)*		
escribir*(A)*		
desbloquear*(A)*		
	bloquear-x*(A)*	
	leer*(A)*	
	escribir*(A)*	
	desbloquear*(A)*	
		bloquear-c*(A)*
		leer*(A)*

Figura 15.8. Planificación parcial con bloqueo de dos fases.

Considérense las dos transacciones siguientes de las que solo se muestran algunas de las operaciones leer y escribir significativas:

T_8: leer(a_1);
leer(a_2);
…
leer(a_n);
escribir(a_1).

T_9: leer(a_1);
leer(a_2);
visualizar($a_1 + a_2$).

Si se emplea el protocolo de bloqueo de dos fases, entonces T_8 debe bloquear a_1 en modo exclusivo. Por tanto, toda ejecución concurrente de ambas transacciones conduce a una ejecución secuencial. Observe sin embargo que T_8 solo necesita el bloqueo en modo exclusivo sobre a_1 al final de su ejecución, cuando escribe a_1. Así, si T_8 pudiera bloquear inicialmente a_1 en modo compartido y después pudiera cambiar el bloqueo a modo exclusivo, se obtendría una mayor concurrencia, ya que T_8 y T_9 podrían acceder a a_1 y a_2 simultáneamente.

Esta observación lleva a un refinamiento del protocolo de bloqueo de dos fases básico, en el cual se permiten **conversiones de bloqueo.** Se va a proporcionar un mecanismo para cambiar un bloqueo compartido por un bloqueo exclusivo y un bloqueo exclusivo por uno compartido. Se denota la conversión del modo compartido al modo exclusivo como **subir,** y la conversión del modo exclusivo al modo compartido como **bajar.** No se puede permitir la conversión de modos arbitrariamente. Por el contrario, la subida solo puede tener lugar en la fase de crecimiento, mientras que la bajada solo puede tener lugar en la fase de decrecimiento.

Volviendo al ejemplo, las transacciones T_8 y T_9 se pueden ejecutar concurrentemente bajo el protocolo de bloqueo de dos fases refinado, como se muestra en la planificación incompleta de la Figura 15.9, en la cual solo se muestran algunas de las operaciones de bloqueo.

Obsérvese que se puede forzar a esperar a una transacción que intente subir un bloqueo sobre un elemento Q. Esta espera forzada tiene lugar si Q está bloqueado actualmente por *otra* transacción en modo compartido.

Del mismo modo que el protocolo de bloqueo de dos fases básico, el bloqueo de dos fases con conversión de bloqueos solo genera planificaciones secuenciables en cuanto a conflictos, y las transacciones se pueden secuenciar según sus puntos de bloqueo. Además, si se mantienen los bloqueos hasta el final de la transacción, las planificaciones son sin cascada.

Para un conjunto de transacciones puede haber planificaciones secuenciables en cuanto a conflictos que no se puedan obtener por medio del protocolo de bloqueo de dos fases. Sin embargo, para obtener planificaciones secuenciables en cuanto a conflictos por medio de protocolos de bloqueo que no sean de dos fases es necesario: o bien tener información adicional de las transacciones, o bien imponer alguna estructura u orden en el conjunto de elementos de datos de la base de datos. Se verán ejemplos cuando se consideren otros protocolos de bloqueo más adelante en este capítulo.

El bloqueo estricto de dos fases y el bloqueo riguroso de dos fases (con conversión de bloqueos) se usan ampliamente en sistemas de bases de datos comerciales.

A continuación se presenta un esquema simple, pero de uso extendido, que genera automáticamente las instrucciones apropiadas de bloqueo y desbloqueo para una transacción, basándose en peticiones de lectura y escritura desde la transacción:

- Cuando una transacción T_i realiza una operación leer*(Q)*, el sistema genera una instrucción bloquear-c*(Q)* seguida de una instrucción leer*(Q)*.
- Cuando T_i realiza una operación escribir*(Q)*, el sistema comprueba si T_i posee ya un bloqueo en modo compartido sobre Q. Si es así, entonces el sistema genera una instrucción subir*(Q)* seguida de la instrucción escribir*(Q)*. En caso contrario, el sistema genera una instrucción bloquear-x*(Q)* seguida de la instrucción escribir*(Q)*.
- Todos los bloqueos que obtenga una transacción no se desbloquean hasta que dicha transacción se comprometa o aborte.

T_8	T_9
bloquear-c(a_1)	
	bloquear-c(a_1)
bloquear-c(a_2)	
	bloquear-c(a_2)
bloquear-c(a_3)	
bloquear-c(a_4)	
	desbloquear(a_1)
	desbloquear(a_2)
bloquear-c(a_n)	
subir(a_1)	

Figura 15.9. Planificación incompleta con conversión de bloqueos.

15.1.4. Implementación de los bloqueos

Un **gestor de bloqueos** se puede implementar como un proceso que recibe mensajes de las transacciones y envía mensajes como respuesta. El proceso gestor de bloqueos responde a los mensajes de solicitud de bloqueo con mensajes de concesión de bloqueo, o con mensajes que solicitan un retroceso de la transacción (en caso de interbloqueos). Los mensajes de desbloqueo solo requieren como respuesta un reconocimiento, pero pueden dar lugar a un mensaje de concesión para otra transacción que esté esperando.

El gestor de bloqueos utiliza la siguiente estructura de datos: para cada elemento de datos que está actualmente bloqueado, mantiene una lista enlazada de registros, uno para cada solicitud, en el orden en el que llegaron las solicitudes. Utiliza una tabla de asociación, indexada por el nombre del elemento de datos, para encontrar la lista enlazada (si la hay) para cada elemento de datos; esta tabla se denomina **tabla de bloqueos.** Cada registro de la lista enlazada de cada elemento de datos anota qué transacción hizo la solicitud, y qué modo de bloqueo se solicitó. El registro también anota si la solicitud ya se ha concedido.

La Figura 15.10 muestra el ejemplo de una tabla de bloqueos. La tabla contiene bloqueos para cinco elementos de datos distintos: I4, I7, I23, I44 e I912. La tabla de bloqueos utiliza cadenas de desbordamiento, de forma que hay una lista enlazada de elementos de datos por cada entrada de la tabla de bloqueos. También hay una lista para cada uno de los elementos de datos, de las transacciones a las que se les han concedido bloqueos, o que están esperando por ellos. Los bloqueos concedidos se han representado como rectángulos rellenos (oscuros), mientras que las solicitudes en espera son los rectángulos vacíos. Se ha omitido el modo de bloqueo para que la figura sea más sencilla. Por ejemplo, se puede observar que a T23 se le han concedido bloqueos sobre I912 e I7, y está esperando para bloquear I4.

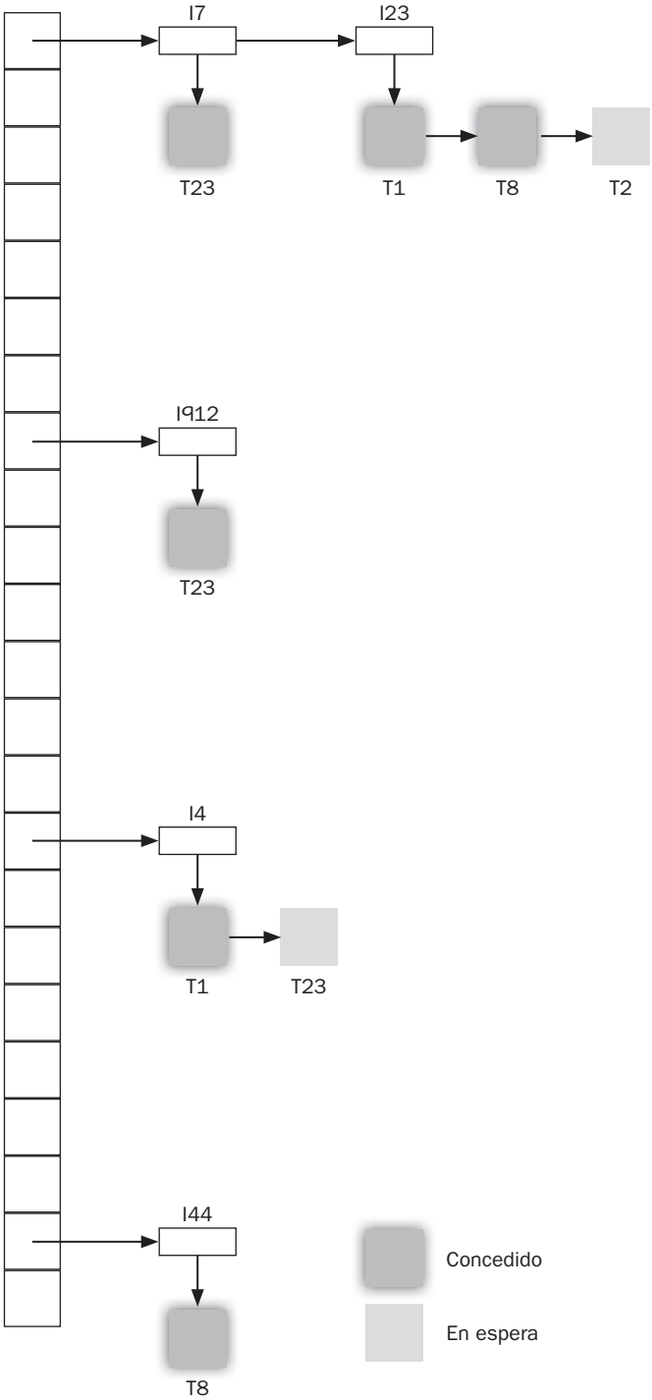


Figura 15.10. Tabla de bloqueos.

Aunque la figura no lo muestre, la tabla de bloqueos debería mantener también un índice de identificadores de transacciones, de forma que fuese posible determinar de manera eficiente el conjunto de bloqueos que mantiene una transacción dada.

El gestor de bloqueos procesa las solicitudes de la siguiente forma:

- Cuando llega un mensaje de solicitud, añade un registro al final de la lista enlazada del elemento de datos; si la lista enlazada existe. En otro caso crea una nueva lista enlazada que tan solo contiene el registro correspondiente a la solicitud.

  Siempre concede la primera solicitud de bloqueo sobre el elemento de datos. Pero si la transacción solicita un bloqueo sobre un elemento sobre el cual ya se ha concedido un bloqueo, el gestor de bloqueos solo concede la solicitud si es compatible con las solicitudes anteriores, y todas estas ya se han concedido. En otro caso, la solicitud tiene que esperar.
- Cuando el gestor de bloqueos recibe un mensaje de desbloqueo de una transacción, borra el registro para ese elemento de datos de la lista enlazada correspondiente a dicha transacción. Comprueba la solicitud del siguiente registro, si existe, como se describe en el párrafo anterior, para determinar si ahora se puede conceder dicha solicitud. Si se puede, el gestor de bloqueos concede la solicitud y procesa el siguiente registro, si lo hay, de forma similar, y así sucesivamente.
- Si una transacción se interrumpe, el gestor de bloqueos borra cualquier solicitud en espera realizada por la transacción. Una vez que el sistema de base de datos ha realizado las acciones apropiadas para deshacer la transacción (véase la Sección 16.3), libera todos los bloqueos que mantenía la transacción abortada.

Este algoritmo garantiza que las solicitudes de bloqueo estén libres de inanición, dado que nunca se puede conceder una solicitud mientras no se hayan concedido las solicitudes recibidas anteriormente. Más adelante, en la Sección 15.2.2, se estudiará cómo detectar y manejar interbloqueos. La Sección 17.2.1 describe una implementación alternativa que utiliza memoria compartida en vez de paso de mensajes para la solicitud y concesión de bloqueos.

15.1.5. Protocolos basados en grafos

Como se ha visto en la Sección 15.1.3, si se desean desarrollar protocolos que no sean de dos fases se necesita información adicional acerca de la forma en que las transacciones acceden a la base de datos. Pueden ofrecer la información adicional de varios modelos, que difieren en la cantidad de información que proporcionan. El modelo más simple exige que se tenga un conocimiento previo acerca del orden en el cual se accede a los elementos de la base de datos. Dada esta información, es posible construir protocolos de bloqueo que no sean de dos fases pero que, no obstante, aseguren la secuencialidad.

Para adquirir tal conocimiento previo, se impone un orden parcial $\rightarrow$ sobre el conjunto $\mathbf{D} = \{d_1, d_2 \ldots , d_h\}$ de todos los elementos de datos. Si $d_i \rightarrow d_j$ entonces toda transacción que acceda tanto a d_i como a d_j debe acceder a d_i antes de acceder a d_j. Este orden parcial puede ser el resultado de la organización tanto lógica como física de los datos, o se puede imponer únicamente por motivos de control de concurrencia.

El orden parcial implica que el conjunto **D** se pueda ver como un grafo dirigido acíclico denominado **grafo de la base de datos.** Para simplificar, esta sección solo se centra en aquellos grafos que son árboles con raíz. Se va a mostrar un protocolo simple llamado *protocolo de árbol*, que está restringido a utilizar solo bloqueos *exclusivos*. En las notas bibliográficas se proporcionan referencias a otros protocolos basados en grafos más complejos.

En el **protocolo de árbol** solo se permite la instrucción de bloqueo bloquear-c. Cada transacción T_i puede bloquear un elemento de datos como mucho una vez, y debe seguir las siguientes reglas:

1. El primer bloqueo de T_i puede ser sobre cualquier elemento de datos.
2. Posteriormente, T_i puede bloquear un elemento de datos Q solo si T_i está bloqueando actualmente al padre de Q.
3. Los elementos de datos se pueden desbloquear en cualquier momento.
4. T_i no puede bloquear de nuevo un elemento de datos que ya haya bloqueado y desbloqueado anteriormente.

Todas las planificaciones que sean legales bajo el protocolo de árbol son secuenciables en cuanto a conflictos.

Para ilustrar este protocolo considere el grafo de la base de datos de la Figura 15.11. Las cuatro transacciones siguientes siguen el protocolo de árbol sobre dicho grafo. Solo se muestran las instrucciones de bloqueo y desbloqueo:

T_{10}: bloquear-x*(B)*; bloquear-x*(E)*; bloquear-x*(D)*; desbloquear*(B)*; desbloquear*(E)*; bloquear-x*(G)*; desbloquear*(D)*; desbloquear*(G)*.

T_{11}: bloquear-x*(D)*; bloquear-x*(H)*; desbloquear*(D)*; desbloquear*(H)*.

T_{12}: bloquear-x*(B)*; bloquear-x*(E)*; desbloquear*(E)*; desbloquear*(B)*.

T_{13}: bloquear-x*(D)*; bloquear-x*(H)*; desbloquear*(D)*; desbloquear*(H)*.

En la Figura 15.12 se muestra una posible planificación en la que participan estas cuatro transacciones. Observe que durante su ejecución, la transacción T_{10} realiza bloqueos sobre dos subárboles *disjuntos*.

Observe que la planificación de la Figura 15.12 es secuenciable en cuanto a conflictos. Se puede demostrar que el protocolo de árbol no solo asegura la secuencialidad en cuanto a conflictos, sino que también asegura la ausencia de interbloqueos.

El protocolo de árbol de la Figura 15.12 no asegura la recuperabilidad y la ausencia de cascadas. Para asegurar la recuperabilidad y la ausencia de cascadas, el protocolo se puede modificar para que no permita liberar bloqueos exclusivos hasta el final de la transacción. Mantener los bloqueos exclusivos hasta el final de la transacción reduce la concurrencia. Existe una alternativa que mejora la concurrencia, pero solo asegura la recuperabilidad: para cada elemento de datos con una escritura no comprometida se registra qué transacción realizó la última escritura sobre el elemento de datos. Cada vez que una transacción T_i realiza una lectura de un elemento de datos no comprometido, se registra una **dependencia de compromiso** de T_i en la transacción que realizó la última escritura sobre el elemento de datos. Entonces, no se permite que la transacción T_i se comprometa hasta que todas las transacciones sobre las que tiene una dependencia de compromiso se comprometan. Si alguna de estas transacciones aborta, también hay que abortar T_i.

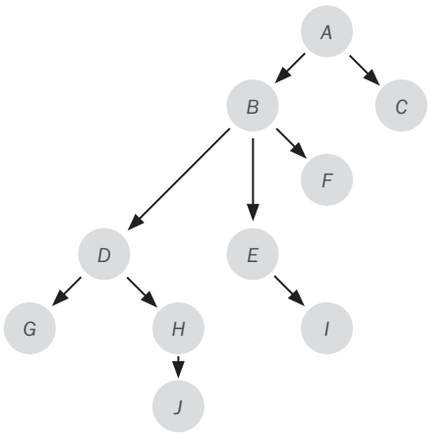


Figura 15.11. Grafo de la base de datos con estructura de árbol.

T_{10}	T_{11}	T_{12}	T_{13}
bloquear-x*(B)*			
	bloquear-x*(D)*		
	bloquear-x*(H)*		
	desbloquear*(D)*		
bloquear-x*(E)*			
bloquear-x*(D)*			
desbloquear*(B)*			
desbloquear*(E)*			
		bloquear-x*(B)*	
		bloquear-x*(E)*	
	desbloquear*(H)*		
bloquear-x*(G)*			
desbloquear*(D)*			
			bloquear-x*(D)*
			bloquear-x*(H)*
			desbloquear*(D)*
			desbloquear*(H)*
		desbloquear*(E)*	
		desbloquear*(B)*	
desbloquear*(G)*			

Figura 15.12. Planificación secuenciable bajo el protocolo de árbol.

El protocolo de bloqueo de árbol tiene la ventaja sobre el protocolo de bloqueo de dos fases de que, a diferencia del bloqueo de dos fases, está libre de interbloqueos, así que no se necesitan retrocesos. El protocolo de bloqueo de árbol tiene otra ventaja sobre el protocolo de bloqueo de dos fases: los desbloqueos se pueden producir antes. Desbloquear antes permite conseguir unos tiempos de espera menores y un aumento de la concurrencia.

Sin embargo, el protocolo tiene el inconveniente de que, en algunos casos, una transacción puede que tenga que bloquear elementos de datos a los que no accede. Por ejemplo, una transacción que tenga que acceder a los elementos de datos A y J en el grafo de la base de datos de la Figura 15.11 debe bloquear no solo A y J, sino también los elementos de datos B, D y H. Estos bloqueos adicionales producen un aumento del coste de los bloqueos, la posibilidad de tiempos de espera adicionales y un descenso potencial de la concurrencia. Además, sin un conocimiento previo de los elementos de datos que es necesario bloquear, las transacciones tienen que bloquear la raíz del árbol y esto puede reducir considerablemente la concurrencia.

Para un conjunto de transacciones pueden existir planificaciones secuenciables en cuanto a conflictos que no se pueden obtener por medio del protocolo de árbol. De hecho, hay planificaciones que son posibles mediante el protocolo de bloqueo de dos fases que no son posibles mediante el protocolo de árbol, y viceversa. En los ejercicios se exponen ejemplos de tales planificaciones.

15.2. Tratamiento de interbloqueos

Un sistema está en estado de interbloqueo si existe un conjunto de transacciones tal que toda transacción del conjunto está esperando a otra transacción del conjunto. Con mayor precisión, existe un conjunto de transacciones en espera $\{T_0, T_1 \ldots, T_n\}$ tal que T_0 está esperando a un elemento de datos que posee T_1, y T_1 está esperando a un elemento de datos que posee T_2, y T_{n-1} está esperando a un elemento de datos que posee T_n, y T_n está esperando a un elemento que posee T_0. En tal situación ninguna de las transacciones puede progresar.

El único remedio a esta situación no deseada es que el sistema realice alguna acción drástica, como hacer retroceder alguna de las transacciones involucradas en el interbloqueo. El retroceso de una transacción puede ser parcial: esto es, se puede retroceder una transacción hasta el punto en que obtuvo un bloqueo cuya liberación resuelve el interbloqueo.

Existen dos métodos principales para tratar el problema de los interbloqueos. Se puede utilizar un protocolo de **prevención de interbloqueos** para asegurar que el sistema *nunca* llega a un estado de interbloqueo. De forma alternativa, se puede permitir que el sistema llegue a un estado de interbloqueo, y tratar de recuperarse después a través de un esquema de **detección de interbloqueo** y **recuperación de interbloqueo.** Como se verá a continuación, ambos pueden provocar un retroceso de las transacciones. La prevención normalmente se usa cuando la probabilidad de que el sistema llegue a un estado de interbloqueo es relativamente alta; en caso contrario, es más eficiente usar la detección y recuperación.

Observe que el esquema de detección y recuperación necesita una sobrecarga que incluye no solo el coste en tiempo de ejecución para mantener la información necesaria para ejecutar el algoritmo de detección, sino también las pérdidas potenciales inherentes a la recuperación de un interbloqueo.

15.2.1. Prevención de interbloqueos

Existen dos enfoques en la prevención de interbloqueos. Un enfoque asegura que no puede haber esperas cíclicas ordenando las peticiones de bloqueo o exigiendo que todos los bloqueos se adquieran juntos. El otro enfoque es más cercano a la recuperación de interbloqueos y realiza retrocesos de las transacciones en lugar de esperar un bloqueo, siempre que el bloqueo pueda llevar potencialmente a un interbloqueo.

El esquema más simple para la primera aproximación exige que cada transacción bloquee todos sus elementos de datos antes de comenzar su ejecución. Además, o bien se bloquean todos en un paso o no se bloquea ninguno de ellos. Este protocolo tiene dos inconvenientes principales: (1) a menudo es difícil predecir, antes de que comience la transacción, cuáles son los elementos de datos que es necesario bloquear; (2) la utilización de elementos de datos puede ser muy baja, ya que muchos de los elementos de datos pueden estar bloqueados pero sin usar durante mucho tiempo.

Otro esquema para prevenir interbloqueos consiste en imponer un orden a todos los elementos de datos y exigir que una transacción bloquee un elemento de datos solo en el orden que especifica dicho orden. Se ha visto un esquema así en el protocolo de árbol, que usa una ordenación parcial de los elementos de datos.

Una variante a esta aproximación es utilizar un orden total de los elementos de datos, en conjunción con el bloqueo de dos fases. Una vez que una transacción ha bloqueado un elemento en particular, no puede solicitar bloqueos sobre elementos que preceden a dicho elemento en la ordenación. Este esquema es fácil de implementar siempre que el conjunto de los elementos de datos a los que accede una transacción se conozca cuando la transacción comienza la ejecución. No hay necesidad de cambiar el sistema de control de concurrencia subyacente si se utiliza un bloqueo de dos fases: todo lo que hace falta es asegurar que los bloqueos se solicitan en el orden adecuado.

La segunda aproximación para prevenir interbloqueos consiste en utilizar expropiación y retrocesos de transacciones. En la expropiación, cuando la transacción T_j solicita un bloqueo que la transacción T_i posee, el bloqueo concedido a T_i puede **expropiarse** retrocediendo T_i y concediéndolo a continuación a T_j. Para controlar la expropiación se asigna una marca temporal única a cada transacción, basada en un contador o en un reloj del sistema. El sistema usa estas marcas temporales solo para decidir si una transacción debe esperar o retroceder. Para el control de concurrencia se sigue usando el bloqueo. Si una transacción retrocede mantiene su marca temporal *antigua* cuando vuelva a comenzar. Se han propuesto dos esquemas de prevención de interbloqueos que usan marcas temporales:

1. El esquema **esperar-morir** está basado en una técnica sin expropiación. Cuando la transacción T_i solicita un elemento de datos que posee actualmente T_j, T_i puede esperar solo si tiene una marca temporal más pequeña que la de T_j (es decir, T_i es anterior a T_j). En caso contrario, T_i retrocede (muere).

   Por ejemplo, suponga que las transacciones T_{14}, T_{15} y T_{16} tienen marcas temporales 5, 10 y 15, respectivamente. Si T_{14} solicita un elemento de datos que posee T_{15}, entonces T_{14} tiene que esperar. Si T_{14} solicita un elemento de datos que posee T_{15}, entonces T_{16} retrocede.
2. El esquema **herir-esperar** está basado en una técnica de expropiación. Es el opuesto al esquema esperar-morir. Cuando la transacción T_i solicita un elemento de datos que posee actualmente T_j, T_i puede esperar solo si tiene una marca temporal mayor que la de T_j (es decir, T_i es más reciente que T_j). En caso contrario, T_j retrocede (T_i hiere a T_j).

   Volviendo al ejemplo anterior con las transacciones T_{14}, T_{15} y T_{16}, si T_{14} solicita un elemento de datos que posee T_{15}, entonces se expropia este elemento de datos y T_{15} retrocede. Si T_{16} solicita el elemento de datos que posee T_{15}, entonces T_{16} debe esperar.

El principal problema de cualquiera de estos dos esquemas es que se pueden dar retrocesos innecesarios.

Otro enfoque simple al tratamiento de interbloqueos se basa en **bloqueos con límite de tiempo.** En este enfoque una transacción que haya solicitado un bloqueo espera como mucho durante un intervalo de tiempo especificado. Si no se concede el bloqueo en ese tiempo, se dice que la transacción está fuera de tiempo y ella misma retrocede y vuelve a comenzar. Si de hecho había un interbloqueo, una o varias de las transacciones involucradas en dicho interbloqueo estarán fuera de tiempo y retrocederán, lo que permitirá continuar a las demás. Este esquema se encuentra en un lugar entre la prevención de interbloqueos, donde nunca puede haber un interbloqueo, y la detección y recuperación, las cuales se describen en la Sección 15.2.2.

El esquema de límite de tiempo es particularmente fácil de implementar y funciona bien si las transacciones son cortas y, si son largas, las esperas deben ser a causa de los interbloqueos. Sin embargo, es difícil en general decidir cuánto debe esperar una transacción para que esté fuera de tiempo. Esperas demasiado largas provocan retardos innecesarios una vez que se ha producido un interbloqueo. Esperas demasiado cortas producen el retroceso de la transacción incluso si no hay interbloqueo, provocando un gasto de recursos. La inanición es también posible con este esquema. Por tanto, el esquema basado en un límite de tiempo tiene una aplicación limitada.

15.2.2. Detección y recuperación de interbloqueos

Si el sistema no utiliza algún protocolo que asegure la ausencia de interbloqueos, entonces se debe usar un esquema de detección y recuperación. Periódicamente se invoca a un algoritmo que examina el estado del sistema para determinar si hay un interbloqueo. Si lo hay, entonces el sistema debe intentar recuperarse del interbloqueo. Para ello, el sistema debe:

- Mantener información sobre la asignación de los elementos de datos a las transacciones, así como de toda petición de elemento de datos pendiente.
- Proporcionar un algoritmo que use esta información para determinar si el sistema ha entrado en un estado de interbloqueo.
- Recuperarse del interbloqueo cuando el algoritmo de detección determine que existe un interbloqueo.

  En esta sección se van a desarrollar estos puntos.

15.2.2.1. Detección de interbloqueos

Los interbloqueos se pueden describir con precisión por medio de un grafo dirigido llamado **grafo de espera.** Este grafo consiste en un par $G = (V, A)$, siendo V el conjunto de vértices y A el conjunto de aristas. El conjunto de vértices consiste en todas las transacciones del sistema. Cada elemento del conjunto A de aristas es un par ordenado $T_i \rightarrow T_j$. Si $T_i \rightarrow T_j$ pertenece a A, entonces hay una arista dirigida de la transacción T_i a T_j, lo cual implica que la transacción T_i está esperando a que la transacción T_j libere un elemento de datos que necesita.

Cuando la transacción T_i solicita un elemento de datos que posee actualmente la transacción T_j, entonces se inserta la arista $T_i \rightarrow T_j$ en el grafo de espera. Esta arista solo se borra cuando la transacción T_j deja de poseer un elemento de datos que necesite la transacción T_i.

Existe un interbloqueo en el sistema si, y solo si, el grafo de espera contiene un ciclo. Se dice que toda transacción involucrada en el ciclo está en interbloqueo. Para detectar los interbloqueos, el sistema debe mantener el grafo de espera e invocar periódicamente a un algoritmo que busque un ciclo en el grafo.

Para ilustrar estos conceptos considere el grafo de espera de la Figura 15.13, que describe la siguiente situación:

- La transacción T_{17} está esperando a las transacciones T_{18} y T_{19}.
- La transacción T_{19} está esperando a la transacción T_{18}.
- La transacción T_{18} está esperando a la transacción T_{20}.

Puesto que el grafo no tiene ciclos, el sistema no está en un estado de interbloqueo.

Suponga ahora que la transacción T_{20} solicita un elemento que posee T_{19}. Se añade la arista $T_{20} \rightarrow T_{19}$ al grafo de espera, lo que resulta en el nuevo estado que se ilustra en la Figura 15.14. Ahora el grafo tiene el ciclo:

$$T_{18} \rightarrow T_{20} \rightarrow T_{19} \rightarrow T_{18}$$

lo que implica que las transacciones T_{18}, T_{19} y T_{20} están en interbloqueo.

Por consiguiente surge la pregunta: ¿cuándo se debe invocar al algoritmo de detección? La respuesta depende de dos factores:

1. ¿Cada cuánto tiempo tiene lugar un interbloqueo?
2. ¿Cuántas transacciones se verán afectadas por el interbloqueo?

Si los interbloqueos se producen con frecuencia, entonces se debe invocar al algoritmo de detección con mayor frecuencia de la usual. Los elementos de datos asignados a las transacciones en interbloqueo no estarán disponibles para otras transacciones hasta que pueda romperse el interbloqueo. Adicionalmente también puede aumentar el número de ciclos en el grafo. En el peor de los casos se invocaría al algoritmo de detección cada vez que una petición de asignación no se pudiera conceder inmediatamente.

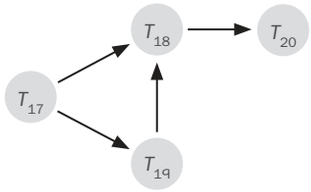


Figura 15.13. Grafo de espera sin ciclos.

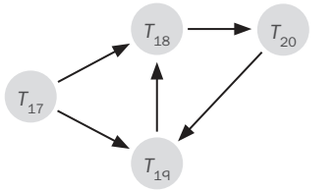


Figura 15.14. Grafo de espera con un ciclo.

15.2.2.2. Recuperación de interbloqueos

Cuando un algoritmo de detección de interbloqueos determina que existe un interbloqueo, el sistema debe **recuperarse** del mismo. La solución más común es retroceder una o más transacciones para romper el interbloqueo. Se deben realizar tres acciones:

1. **Selección de una víctima.** Dado un conjunto de transacciones en interbloqueo, se debe determinar la transacción (o transacciones) que se van a retroceder para romper el interbloqueo. Se deben retroceder aquellas transacciones que incurran en un coste mínimo. Desafortunadamente, el término *coste mínimo* no es nada preciso. Hay muchos factores que determinan el coste de un retroceso, como:
   a. El cómputo que lleva la transacción hasta entonces y lo que le queda todavía hasta completar la tarea asignada.
   b. El número de elementos de datos que ha usado la transacción.
   c. La cantidad de elementos de datos que necesita la transacción para que esta se complete.
   d. El número de transacciones que se verán involucradas en el retroceso.
2. **Retroceso.** Una vez se ha decidido que retrocederá una transacción en particular, se debe determinar hasta dónde retrocederá dicha transacción.

   La solución más simple consiste en un **retroceso total:** se aborta la transacción y luego vuelve a comenzar. Sin embargo, es más efectivo hacer retroceder la transacción solo lo necesario para romper el interbloqueo. Dicho **retroceso parcial** requiere que el sistema mantenga información adicional sobre el estado de todas las transacciones que están en ejecución. Concretamente, es necesario registrar la secuencia de solicitudes/concesiones de bloqueos y de actualizaciones realizadas por la transacción. El mecanismo de detección de interbloqueos debería decidir qué bloqueos necesita liberar la transacción seleccionada para romper el interbloqueo. Hay que hacer retroceder la transacción seleccionada hasta el punto donde obtuvo el primero de esos bloqueos, deshaciendo todas las acciones que realizó desde ese punto. El mecanismo de recuperación debe ser capaz de realizar tales retrocesos parciales. Además, las transacciones deben poder reanudar la ejecución después de un retroceso parcial. Consulte las notas bibliográficas para obtener más referencias.
3. **Inanición.** En un sistema en el cual la selección de víctimas esté basada principalmente en factores de coste, puede ocurrir que siempre se elija a la misma transacción como víctima. El resultado es que esta transacción no completa nunca su tarea. Dicha situación se denomina **inanición.** Se debe asegurar que una transacción pueda elegirse como víctima solo un número finito (y pequeño) de veces. La solución más común consiste en incluir en el factor de coste el número de retrocesos.

15.3. Granularidad múltiple

En los esquemas de control de concurrencia descritos hasta ahora, se han usado los elementos de datos individuales como la unidad sobre la cual se producía la sincronización.

Sin embargo, hay circunstancias en las que puede ser conveniente agrupar varios elementos de datos y tratarlos como una unidad individual de sincronización. Por ejemplo, si la transacción T_i tiene que acceder a toda la base de datos y se usa un protocolo de bloqueo, entonces T_i debe bloquear cada elemento de la base de datos. Ejecutar estos bloqueos produce claramente un consumo de tiempo. Sería mejor que T_i pudiera realizar una *única* petición de bloqueo para bloquear toda la base de datos. Por otro lado, si la transacción T_j necesita acceder solo a unos cuantos datos no sería necesario bloquear toda la base de datos, ya que en ese caso se perdería la concurrencia.

Lo que se necesita es un mecanismo que permita al sistema definir múltiples niveles de **granularidad.** Se puede lograr permitiendo que los elementos de datos sean de varios tamaños y definiendo una jerarquía de granularidades de los datos, en la cual las granularidades pequeñas están anidadas en otras más grandes. Esta jerarquía se puede representar gráficamente como un árbol. Observe que este árbol es significativamente distinto del que se usaba en el protocolo de árbol (Sección 15.1.5). Un nodo interno del árbol de granularidad múltiple representa los datos que se asocian con sus descendientes. En el protocolo de árbol, cada nodo es un elemento de datos independiente.

Como ejemplo considere el árbol de la Figura 15.15, que consta de cuatro niveles de nodos. El nivel superior representa toda la base de datos. Debajo están los nodos de tipo *zona*; la base de datos consta exactamente de estas zonas. Cada zona tiene nodos de tipo *archivo* como hijos. Cada zona contiene exactamente aquellos archivos que sean sus nodos hijos. Ningún archivo está en más de una zona. Finalmente, cada archivo tiene nodos de tipo *registro*. Como antes, un archivo consta exactamente de aquellos registros que sean sus nodos hijos y ningún registro puede estar presente en más de un archivo.

Se puede bloquear individualmente cada nodo del árbol. Como ya se hizo en el protocolo de bloqueo de dos fases, se van a usar los modos de bloqueo **compartido** y **exclusivo.** Cuando una transacción bloquea un nodo, en modo tanto compartido como exclusivo, también bloquea todos los descendientes de ese nodo con el mismo modo de bloqueo. Por ejemplo, si la transacción T_i **bloquea explícitamente** el archivo A_c de la Figura 15.15 en modo exclusivo, entonces **bloquea implícitamente** en modo exclusivo todos los registros que pertenecen a dicho archivo. No es necesario que bloquee explícitamente cada registro individual de A_c.

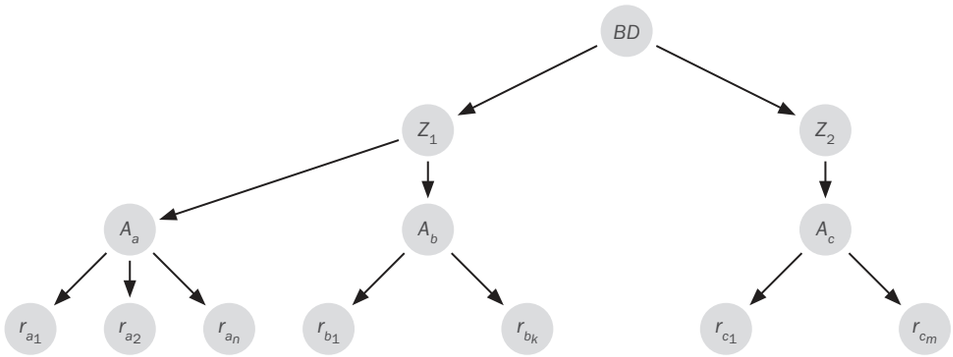


Figura 15.15. Jerarquía de granularidad.

Suponga que la transacción T_j quiere bloquear el registro r_{b6} del archivo A_b. Puesto que T_i ha bloqueado A_b explícitamente, se entiende que también se bloquea r_{b6} (implícitamente). Pero cuando T_j realiza una petición de bloqueo sobre r_{b6}, r_{b6} no está bloqueado explícitamente ¿Cómo puede determinar el sistema si T_j puede bloquear r_{b6}? T_j debe recorrer el árbol desde la raíz hasta el nodo r_{b6}. Si cualquier nodo del camino está bloqueado en un modo incompatible, entonces hay que retrasar T_j.

Suponga ahora que la transacción T_k quiere bloquear toda la base de datos. Para ello debe bloquear simplemente la raíz de la jerarquía. Observe, sin embargo, que T_k no tendrá éxito al bloquear el nodo raíz, ya que T_i posee un bloqueo sobre parte del árbol (en concreto sobre el archivo A_b). ¿Pero cómo determina el sistema si se puede bloquear el nodo raíz? Una posibilidad es buscar en todo el árbol. Sin embargo, esta solución anula el propósito del esquema de bloqueo de granularidad múltiple. Un modo más eficiente de obtener este conocimiento es introducir una nueva clase de modo de bloqueo que se denomina **modo de bloqueo intencional.** Si un nodo se bloquea en modo intencional, se está haciendo un bloqueo explícito en un nivel inferior del árbol (es decir, en una granularidad más fina). Los bloqueos intencionales se colocan en todos los ascendientes de un nodo antes de bloquearlo explícitamente. Así, no es necesario que una transacción busque en todo el árbol para determinar si puede bloquear un nodo con éxito. Una transacción que quiera bloquear un nodo — por ejemplo Q— debe recorrer el camino en el árbol desde la raíz hasta Q. Durante el recorrido del árbol la transacción bloquea los distintos nodos en un modo intencional.

Hay un modo intencional asociado al modo compartido y hay otro asociado al modo exclusivo. Si un nodo se bloquea en **modo intencional-compartido (IC),** se realiza un bloqueo explícito en un nivel inferior del árbol, pero solo con bloqueos en modo compartido. De forma similar, si se bloquea un nodo en **modo intencional-exclusivo (IX),** entonces el bloqueo explícito se hace en un nivel inferior con bloqueos en modo exclusivo o en modo compartido. Finalmente, si se bloquea un nodo en **modo intencional-exclusivo y compartido (IXC),** el subárbol cuya raíz es ese nodo se bloquea explícitamente en modo exclusivo, y dicho bloqueo explícito se produce en un nivel inferior con bloqueos en modo exclusivo. La función de compatibilidad para estos modos de bloqueo se presenta en la Figura 15.16.

El **protocolo de bloqueo de granularidad múltiple** usa estos bloqueos para asegurar la secuencialidad. Requiere que cada transacción T_i que intente bloquear un nodo Q siga las siguientes reglas:

1. La transacción T_i debe observar la función de compatibilidad de bloqueos de la Figura 15.16.
2. La transacción T_i debe bloquear la raíz del árbol en primer lugar y puede hacerlo en cualquier modo.
3. La transacción T_i puede bloquear un nodo Q en el modo C o IC solo si actualmente T_i está bloqueando al padre de Q en el modo IX o IC.
4. La transacción T_i puede bloquear un nodo Q en el modo X, IXC o IX solo si actualmente T_i está bloqueando al padre de Q en el modo IX o IXC.
5. La transacción T_i puede bloquear un nodo solo si T_i no ha desbloqueado previamente ningún nodo (es decir, T_i es de dos fases).
6. La transacción T_i puede desbloquear un nodo Q solo si actualmente T_i no ha bloqueado a ninguno de los hijos de Q.

	IC	IX	C	IXC	X
IC	cierto	cierto	cierto	cierto	falso
IX	cierto	cierto	falso	falso	falso
C	cierto	falso	cierto	falso	falso
IXC	cierto	falso	falso	falso	falso
X	falso	falso	falso	falso	falso

Figura 15.16. Matriz de compatibilidad.

Observe que en el protocolo de granularidad múltiple es necesario que los bloqueos se adquieran en *orden descendente* (de la raíz a las hojas), y que se liberen en *orden ascendente* (de las hojas a la raíz).

Para ilustrar este protocolo considere el árbol de la Figura 15.15 y las siguientes transacciones:

- Suponga que la transacción T_{21} lee el registro r_{a2} del archivo A_a. Entonces T_{21} necesita bloquear la base de datos, la zona Z_1 y A_a en el modo IC (y en este orden) y, finalmente, bloquear r_{a2} en el modo C.
- Suponga que la transacción T_{22} modifica el registro r_{a9} del archivo A_a. Entonces T_{22} necesita bloquear la base de datos, la zona Z_1 y el archivo A_a (en este orden) en el modo IX y, finalmente, bloquear r_{a9} en el modo X.
- Suponga que la transacción T_{23} lee todos los registros del archivo A_a. Entonces T_{23} necesita bloquear la base de datos y la zona Z_1 (en este orden) en el modo IC y, finalmente, bloquear A_a en el modo C.
- Suponga que la transacción T_{24} lee toda la base de datos. Puede hacerlo una vez que bloquee la base de datos en modo C.

Se observa que las transacciones T_{21}, T_{23} y T_{24} pueden acceder concurrentemente a la base de datos. La transacción T_{22} se puede ejecutar concurrentemente con T_{21}, pero no con T_{23} ni con T_{24}.

Este protocolo permite mejorar la concurrencia y reducir la sobrecarga de bloqueos. Es particularmente útil en las aplicaciones con una mezcla de:

- Transacciones cortas que solo acceden a algunos elementos de datos.
- Transacciones largas que producen informes a partir de un archivo completo o un conjunto de archivos.

Existe un protocolo de bloqueo similar que es aplicable a sistemas de bases de datos en los cuales las granularidades de los datos se organizan en forma de grafo dirigido acíclico. Consulte las notas bibliográficas para más referencias. En este protocolo son posibles los interbloqueos, al igual que en los protocolos de bloqueo de dos fases. Existen técnicas para reducir la frecuencia de interbloqueos en el protocolo de granularidad múltiple y también para eliminar completamente los interbloqueos. Se hace referencia a estas técnicas en las notas bibliográficas.

15.4. Protocolos basados en marcas temporales

En los protocolos de bloqueo que se han descrito anteriormente se determina el orden entre dos transacciones conflictivas en tiempo de ejecución a través del primer bloqueo que soliciten ambas con modos incompatibles. Otro método para determinar el orden de secuencialidad es seleccionar previamente un orden entre las transacciones. El método más común para hacerlo es utilizar un esquema de *ordenación por marcas temporales*.

15.4.1. Marcas temporales

A toda transacción T_i del sistema se le asocia una única marca temporal fija, denotada por MT(T_i). El sistema de base de datos asigna esta marca temporal antes de que comience la ejecución de T_i. Si a la transacción T_i se le ha asignado la marca temporal MT(T_i) y una nueva transacción T_j entra en el sistema, entonces MT$(T_i) <$ MT(T_j). Existen dos métodos simples para implementar este esquema:

1. Usar el valor del **reloj del sistema** como marca temporal; es decir, la marca temporal de una transacción es igual al valor del reloj en el momento en el que la transacción entra en el sistema.
2. Usar un **contador lógico** que se incrementa cada vez que se asigna una nueva marca temporal; es decir, la marca temporal de una transacción es igual al valor del contador en el momento en el cual la transacción entra en el sistema.

Las marcas temporales de las transacciones determinan el orden de secuencia. De este modo, si MT$(T_i) <$ MT(T_j), entonces el sistema debe asegurar que toda planificación que produzca sea equivalente a una planificación secuencial en la cual la transacción T_i aparezca antes que la transacción T_j.

Para implementar este esquema, se asocian a cada elemento de datos Q dos valores de marca temporal:

- **Marca-temporal-E***(Q)*: denota la mayor marca temporal de todas las transacciones que ejecutan con éxito escribir*(Q)*.
- **Marca-temporal-L***(Q)*: denota la mayor marca temporal de todas las transacciones que ejecutan con éxito leer*(Q)*.

Estas marcas temporales se actualizan cada vez que se ejecuta una nueva operación leer*(Q)* o escribir*(Q)*.

15.4.2. Protocolo de ordenación por marcas temporales

El **protocolo de ordenación por marcas temporales** asegura que todas las operaciones leer y escribir conflictivas se ejecutan en el orden de las marcas temporales. Este protocolo funciona de la siguiente forma:

1. Suponga que la transacción T_i ejecuta leer*(Q)*.
   a. Si MT$(T_i) <$ marca-temporal-E(Q), entonces T_i necesita leer un valor de Q que ya se haya sobrescrito. Por tanto, se rechaza la operación leer y T_i retrocede.
   b. Si MT$(T_i) \geq$ marca-temporal-E(Q), entonces se ejecuta la operación leer y marca-temporal-L(Q) se asigna al máximo valor de marca-temporal-L(Q) y de MT(T_i).
2. Suponga que la transacción T_i ejecuta escribir*(Q)*.
   a. Si MT*(T_i)* $<$ marca-temporal-L*(Q)*, entonces se necesita previamente el valor de Q que produce T_i y el sistema asume que dicho valor nunca se puede producir. Por tanto, se rechaza la operación escribir y T_i retrocede.
   b. Si MT*(T_i)* $<$ marca-temporal-E*(Q)*, entonces T_i está intentando escribir un valor de Q obsoleto. Por tanto, se rechaza la operación escribir y T_i retrocede.
   c. En otro caso, el sistema ejecuta la operación escribir y se asigna a marca-temporal-E*(Q)* la MT*(T_i)*.

A una transacción T_i que el esquema de control de concurrencia haya retrocedido como resultado de la ejecución de una operación leer o escribir se le asigna una nueva marca temporal y se inicia de nuevo.

Para ilustrar este protocolo considere las transacciones T_{25} y T_{26}. La transacción T_{25} visualiza el contenido de las cuentas A y B:

```
T25: leer(B);
     leer(A);
     visualizar(A + B).
```

La transacción T_{26} transfiere 50 € de la cuenta B a la A, y muestra después el contenido de ambas:

```
T26: leer(B);
     B := B − 50;
     escribir(B);
     leer(A);
     A := A + 50;
     escribir(A);
     visualizar(A + B).
```

En las planificaciones actuales con el protocolo de marcas temporales se supondrá que a una transacción se le asigna una marca temporal inmediatamente antes de su primera instrucción. Así, en la planificación 3 de la Figura 15.17, MT(T_{25}) < MT(T_{26}) y es posible la planificación con el protocolo de marcas temporales.

Observe que la ejecución anterior puede obtenerse también con el protocolo de bloqueo de dos fases. Sin embargo, existen planificaciones que son posibles con el protocolo de bloqueo de dos fases que no lo son con el protocolo de marcas temporales y viceversa (véase el Ejercicio 15.29).

El protocolo de ordenación por marcas temporales asegura la secuencialidad en cuanto a conflictos. Esta afirmación se deduce del hecho de que las operaciones conflictivas se procesan durante la ordenación de las marcas temporales.

El protocolo asegura la ausencia de interbloqueos, ya que ninguna transacción tiene que esperar. Sin embargo, existe una posibilidad de inanición de las transacciones largas si una secuencia de transacciones cortas conflictivas provoca reinicios repetidos de la transacción larga. Si se descubre que una transacción se está reiniciando de forma repetida, es necesario bloquear las transacciones conflictivas de forma temporal para permitir que la transacción termine.

T_{25}	T_{26}
leer(*B*)	
	leer(*B*)
	B := *B* – 50
	escribir(*B*)
leer(*A*)	
	leer(*A*)
visualizar(*A* + *B*)	
	A := *A* + 50
	escribir(*A*)
	visualizar(*A* + *B*)

Figura 15.17. Planificación 3.

El protocolo puede generar planificaciones no recuperables; sin embargo, se puede extender para producir planificaciones recuperables de una de las siguientes formas:

- La recuperabilidad y la ausencia de cascadas se pueden asegurar realizando todas las escrituras juntas al final de la transacción. Las escrituras tienen que ser atómicas en el siguiente sentido: mientras se estén realizando las escrituras, no se permite que ninguna transacción acceda a ninguno de los elementos de datos que se han escrito.
- La recuperabilidad y la ausencia de cascadas también se pueden garantizar utilizando una forma limitada de bloqueo, por medio de la cual las lecturas de los elementos no comprometidos se posponen hasta que la transacción haya actualizado el elemento comprometido (véase el Ejercicio 15.30).
- Solamente se puede asegurar la recuperabilidad realizando un seguimiento de las escrituras no comprometidas y únicamente permitiendo que una transacción T_i se comprometa después de que lo haga cualquier transacción que escribió un valor que leyó T_i. Las dependencias de compromiso, esbozadas en la Sección 15.1.5, se pueden utilizar para este propósito.

15.4.3. Regla de escritura de Thomas

Ahora se presenta una modificación del protocolo de ordenación por marcas temporales que permite una mayor concurrencia potencial que la que tiene el protocolo de la Sección 15.4.2. Considere la planificación 4 de la Figura 15.18 y aplique el protocolo de ordenación por marcas temporales. Puesto que T_{27} comienza antes que T_{28}, se asume que MT(T_{27}) < MT(T_{28}). La operación leer*(Q)* de T_{27} tiene éxito, así como la operación escribir*(Q)* de T_{28}. Cuando T_{27} intenta hacer su operación escribir*(Q)* se encuentra con que MT(T_{27}) < marca-temporal-E*(Q)*, ya que marca-temporal-E*(Q)* = MT(T_{28}). De este modo se rechaza la operación escribir*(Q)* de T_{27} y la transacción T_{27} se debe retroceder.

A pesar de que el protocolo de ordenación por marcas temporales exige el retroceso de la transacción T_{27}, no es necesario. Como T_{28} ya ha escrito *Q*, el valor que intenta escribir T_{27} nunca se va a leer. Toda transacción T_i con MT(T_i) < MT(T_{28}) que intente una operación leer*(Q)* retrocederá, ya que MT(T_i) < marca-temporal-E*(Q)*. Toda transacción T_j con MT(T_j) > MT(T_{28}) debe leer el valor de *Q* que ha escrito T_{28} en lugar del valor que T_{27} intenta escribir.

Esta observación lleva a una versión modificada del protocolo de ordenación por marcas temporales, en el cual se pueden ignorar, bajo ciertas circunstancias, las operaciones escribir obsoletas. Las reglas del protocolo para las operaciones leer no sufren cambios. Sin embargo, las reglas del protocolo para las operaciones escribir son algo diferentes de las del protocolo de ordenación por marcas temporales de la Sección 15.4.2.

La modificación al protocolo de ordenación por marcas temporales, denominada **regla de escritura de Thomas,** es la siguiente: suponga que la transacción T_i ejecuta escribir*(Q)*.

1. Si MT(T_i) < marca-temporal-L*(Q)*, entonces se necesita previamente el valor de *Q* que produce T_i y el sistema asume que dicho valor no se puede producir nunca. Por tanto, se rechaza la operación escribir y T_i retrocede.
2. Si MT(T_i) < marca-temporal-E*(Q)*, entonces T_i está intentando escribir un valor de *Q* obsoleto. Por tanto, se puede ignorar dicha operación escribir.
3. En otro caso se ejecuta la operación escribir y MT(T_i) se asigna a marca-temporal-E*(Q)*.

La diferencia entre las reglas anteriores y las de la Sección 15.4.2 está en la segunda regla. El protocolo de ordenación por marcas temporales exige que T_i retroceda si ejecuta escribir*(Q)* y MT(T_i) < marca-temporal-E*(Q)*. Sin embargo, ahora se ignora la instrucción escribir obsoleta en aquellos casos en los que MT(T_i) ≥ marca-temporal-L*(Q)*.

Al ignorar la escritura, la regla de escritura de Thomas permite planificaciones que no son serializables en cuanto a conflictos, pero que son correctas. Estas planificaciones no serializables en cuanto a conflictos satisfacen la definición de planificaciones *serializables en cuanto a vistas* (véase el cuadro en pág. siguiente). La regla de escritura de Thomas emplea la secuencialidad en cuanto a vistas, borrando las operaciones escribir obsoletas de las transacciones que las ejecutan. Esta modificación de las transacciones hace posible generar planificaciones que no serían posibles con otros protocolos que se han presentado en este capítulo. Por ejemplo, la planificación 4 de la Figura 15.18 no es secuenciable en cuanto a conflictos y, por tanto, no es posible con el protocolo de bloqueo de dos fases, con el protocolo de árbol ni con el protocolo de ordenación por marcas temporales. Con la regla de escritura de Thomas se ignoraría la operación escribir*(Q)* de T_{27}. El resultado es una planificación que es equivalente en cuanto a vistas a la planificación secuencial <T_{27}, T_{28}>.

T_{27}	T_{28}
leer(*Q*)	
	escribir(*Q*)
escribir(*Q*)	

Figura 15.18. Planificación 4.

Secuencialidad de vistas

Existe otra forma de equivalencia menos exigente que la equivalencia en cuanto a conflictos, pero que, como esta, solo se basa en las operaciones leer y escribir de la transacción.

Suponga dos planificaciones S y S', en las que participan el mismo conjunto de planificaciones. Se dice que las planificaciones S y S' son **equivalentes en cuanto a vistas** si se cumplen tres condiciones:

1. Para cada elemento Q, si la transacción T_i lee el valor inicial de Q en la planificación S, entonces T_i también lee el valor inicial de Q en la planificación S'.
2. Para cada elemento Q, si la transacción T_i ejecuta leer*(Q)* en la planificación S, y si ese valor lo produjo una operación escribir*(Q)* de la transacción T_j, entonces la operación leer*(Q)* de la transacción T_i debe leer también en la planificación S' el valor de Q que produjo la misma operación escribir*(Q)* de la transacción T_j.
3. Para cada elemento Q, la transacción que realiza la operación escribir*(Q)* final (si existe) en la planificación S debe realizar la operación escribir*(Q)* final en la planificación S'.

Las condiciones 1 y 2 aseguran que cada transacción lee los mismos valores en ambas planificaciones y, por tanto, realizan los mismos cómputos. La condición 3, junto con las condiciones 1 y 2, aseguran que las dos planificaciones generan el mismo estado final del sistema.

El concepto de equivalencia de vistas conduce al concepto de secuencialidad de vistas. Se dice que una planificación S es **secuenciable en cuanto a vistas** si equivale a una planificación secuencial en cuanto a vistas.

Como ejemplo, suponga que se amplía la planificación 4 con la transacción T_{29}, y se obtiene la siguiente secuencialidad de vistas (planificación 5):

T_{27}	T_{28}	T_{29}
leer*(Q)*		
	escribir*(Q)*	
escribir*(Q)*		
		escribir*(Q)*

De hecho, la planificación 5 es equivalente en cuanto a vistas con la planificación secuencial $<T_{27}, T_{28}, T_{29}>$, ya que la única operación leer*(Q)* lee el valor inicial de Q en ambas planificaciones y T_{29} realiza la escritura final de Q en ambas planificaciones.

Toda planificación secuenciable en cuanto a conflictos también es secuenciable en cuanto a vistas, pero existen planificaciones secuenciables en cuanto a vistas que no lo son en cuanto a conflictos. De hecho, la planificación 5 no es secuenciable en cuanto a conflictos, ya que todos los pares de instrucciones consecutivas entran en conflicto y, por tanto, no se puede realizar un intercambio entre ellas.

Observe que, en la planificación 5, las transacciones T_{28} y T_{29} realizan operaciones escribir*(Q)* sin haber realizado ninguna operación leer*(Q)*. Este tipo de escritura se denomina **escritura ciega.** Aparece en cualquier planificación secuenciable en cuanto a vistas que no es secuenciable en cuanto a conflictos.

15.5. Protocolos basados en validación

En aquellos casos en que la mayoría de las transacciones son solo de lectura, la tasa de conflictos entre las transacciones puede ser baja. Así, muchas de esas transacciones, si se ejecutasen sin la supervisión de un esquema de control de concurrencia, llevarían no obstante al sistema a un estado consistente. Un esquema de control de concurrencia impone una sobrecarga en la ejecución de código y un posible retardo en las transacciones. Sería aconsejable utilizar otro esquema alternativo que impusiera menos sobrecarga. La dificultad de reducir la sobrecarga está en que no se conocen de antemano las transacciones que estarán involucradas en un conflicto. Para obtener dicho conocimiento se necesita un esquema para *supervisar* el sistema.

El **protocolo de validación** requiere que cada transacción T_i se ejecute en dos o tres fases diferentes durante su tiempo de vida, dependiendo de si es una transacción de solo lectura o de actualización. Las fases son, en orden:

1. **Fase de lectura.** Durante esta fase, el sistema ejecuta la transacción T_i. Se leen los valores de varios elementos de datos y se almacenan en variables locales de T_i. Todas las operaciones escribir se realizan sobre las variables locales temporales sin actualizar la base de datos.
2. **Fase de validación.** La transacción T_i realiza una prueba de validación (descrita más adelante) para determinar si puede pasar a la fase de copiar a la base de datos las variables locales temporales que tienen los resultados de las operaciones escribir sin causar una violación de la secuencialidad. Si la transacción no cumple la prueba de validación, el sistema aborta la transacción.
3. **Fase de escritura.** Si la transacción T_i satisface la prueba de validación, entonces las variables locales temporales que guardan los resultados de las operaciones escribir de T_i se aplican a la base de datos. Las transacciones que solo leen omiten este paso.

Todas las transacciones deben pasar por las tres fases y en el orden que se muestra. Sin embargo, en la ejecución concurrente de las transacciones se pueden entrelazar las tres fases.

Para realizar la prueba de validación se necesita conocer el momento en que tienen lugar las distintas fases de las transacciones T_i. Se asociarán por tanto tres marcas temporales distintas a cada transacción T_i:

1. **Inicio***(T_i)*. Momento en el cual T_i comienza su ejecución.
2. **Validación***(T_i)*. Momento en el cual T_i termina su fase de lectura y comienza su fase de validación.
3. **Fin***(T_i)*. Momento en el cual T_i termina su fase de escritura.

Se determina el orden de secuencialidad mediante la técnica de la ordenación por marcas temporales utilizando el valor de la marca temporal Validación(T_i). De este modo, el valor MT(T_i) = Validación(T_i) y si MT(T_j) < MT(T_k), entonces toda planificación que se produzca debe ser equivalente a una planificación secuencial en la que la transacción T_j aparezca antes que la transacción T_k. La razón de elegir Validación(T_i) como marca temporal de la transacción T_i, en lugar de Inicio(T_i), es porque resulta posible esperar un tiempo de respuesta más rápido, dado que la tasa de conflictos entre transacciones es realmente bajo.

La **comprobación de validación** para la transacción T_i exige que, para toda transacción T_k con MT(T_k) < MT(T_i), se cumpla una de las dos condiciones siguientes:

1. Fin(T_k) < Inicio(T_i). Puesto que T_k completa su ejecución antes de que comience T_i, el orden de secuencialidad realmente se mantiene.
2. El conjunto de todos los elementos de datos que escribe T_k tiene intersección vacía con el conjunto de elementos de datos que lee T_i, y T_k completa su fase de escritura antes de que T_i comience su fase de validación [Inicio(T_i) < Fin(T_k) < Validación(T_i)]. Esta condición asegura que las escrituras de T_k y T_i no se superpongan. Puesto que las escrituras de T_k no afectan a la lectura de T_i, y puesto que T_i no puede afectar a la lectura de T_k, el orden de secuencialidad realmente se mantiene.

Como ejemplo, considere de nuevo las transacciones T_{25} y T_{26}. Suponga que MT(T_{25}) < MT(T_{26}). Entonces la fase de validación tiene éxito y produce la planificación 6 de la Figura 15.19. Observe que las escrituras a las variables reales se realizan solo después de la fase de validación de T_{26}. Así, T_{25} lee los valores anteriores de B y A, y la planificación es secuenciable.

El esquema de validación evita automáticamente los retrocesos en cascada, ya que las escrituras reales solo tienen lugar después de que la transacción que ejecuta la escritura se haya comprometido. Sin embargo, existe una posibilidad de inanición de las transacciones largas debido a una secuencia de transacciones cortas conflictivas que provoca reinicios repetidos de la transacción larga. Para evitar la inanición es necesario bloquear las transacciones conflictivas de forma temporal para permitir que la transacción larga termine.

Este esquema de validación se denomina esquema de **control de concurrencia optimista,** dado que las transacciones se ejecutan de forma optimista, asumiendo que serán capaces de finalizar la ejecución y validar al final. En cambio, los bloqueos y la ordenación por marcas temporales son pesimistas en cuanto que fuerzan una espera o un retroceso cada vez que se detecta un conflicto, aun cuando existe una posibilidad de que la planificación sea secuenciable en cuanto a conflictos.

T_{25}	T_{26}
leer(*B*)	
	leer(*B*)
	B := *B* − 50
	leer(*A*)
	A := *A* + 50
leer(*A*)	
<*validar*>	
visualizar(*A* + *B*)	
	<*validar*>
	escribir(*B*)
	escribir(*A*)

Figura 15.19. Planificación 6 producida usando validación.

15.6. Esquemas multiversión

Los esquemas de control de concurrencia que se han descrito anteriormente aseguran la secuencialidad bien retrasando una operación, o bien abortando la transacción que realiza la operación. Por ejemplo, una operación leer se puede retrasar porque todavía no se haya escrito el valor apropiado; o se puede rechazar (es decir, la transacción que la realiza debe abortarse) porque se haya sobrescrito el valor que supuestamente se iba a leer. Se podrían evitar estos problemas si se mantuvieran en el sistema copias anteriores de cada elemento de datos.

En los esquemas de **control de concurrencia multiversión,** cada operación escribir(*Q*) crea una nueva **versión** de Q. Cuando se realiza una operación leer(*Q*), el gestor de control de concurrencia elige una de las versiones de Q que se va a leer. El esquema de control de concurrencia debe asegurar que la elección de la versión que se va a leer se haga de tal manera que asegure la secuencialidad. Así mismo, es crucial, por motivos de rendimiento, que una transacción sea capaz de determinar rápida y fácilmente la versión del elemento de datos que se va a leer.

15.6.1. Ordenación por marcas temporales multiversión

El protocolo de ordenación por marcas temporales se puede extender a un protocolo multiversión. A cada transacción T_i del sistema se le asocia una única marca temporal estática que se denota por MT(T_i). El sistema de bases de dates asigna dicha marca temporal antes de que la transacción comience su ejecución, como se ha descrito en la Sección 15.4.

A cada elemento de datos Q se asocia una secuencia de versiones $<Q_1, Q_2 \ldots , Q_m>$. Cada versión Q_k contiene tres campos de datos:

- **contenido** es el valor de la versión Q_k.
- **marca-temporal-E**(Q_k) es la marca temporal de la transacción que haya creado la versión Q_k .
- **marca-temporal-L**(Q_k) es la mayor marca temporal de todas las transacciones que hayan leído con éxito la versión Q_k.

Una transacción — por ejemplo, T_i — crea una nueva versión Q_k del elemento de datos Q realizando la operación escribir(*Q*). El campo contenido de la versión tiene el valor que ha escrito T_i. El sistema inicializa la marca-temporal-E y la marca-temporal-L con MT(T_i). El valor de marca-temporal-L se actualiza cada vez que una transacción T_j lee el contenido de Q_k y la marca-temporal-L(Q_k) < MT(T_j).

El **esquema de marcas temporales multiversión** que se muestra a continuación asegura la secuencialidad. El esquema opera como sigue. Suponga que la transacción T_i realiza una operación leer(*Q*) o escribir(*Q*). Sea Q_k la versión de Q cuya marca temporal de escritura es la mayor marca temporal menor o igual que MT(T_i).

1. Si la transacción T_i ejecuta leer(*Q*), entonces el valor que se devuelve es el contenido de la versión Q_k.
2. Si la transacción T_i ejecuta escribir(*Q*) y si MT(T_i) < marca-temporal-L(Q_k), entonces la transacción T_i retrocede. Si no, si MT(T_i) = marca-temporal-E(Q_k), se sobrescribe el contenido de Q_k , y en otro caso se crea una nueva versión de Q.

La justificación de la regla 1 es clara. Una transacción lee la versión más reciente que viene antes de ella en el tiempo. La segunda regla fuerza a que se aborte una transacción que realice una escritura «demasiado tarde». Con más precisión, si T_i intenta escribir una versión que alguna otra transacción haya leído, entonces no se puede permitir que dicha escritura se realice.

Las versiones que ya no se necesitan se borran basándose en la siguiente regla. Suponga que hay dos versiones Q_k y Q_j, de un elemento de datos y que ambas tienen marca-temporal-E menor que la marca temporal de la transacción más antigua en el sistema. Entonces no se volverá a usar la versión más antigua de Q_k y Q_j y se puede borrar.

El esquema de ordenación por marcas temporales multiversión tiene la propiedad deseable de que nunca falla una petición de lectura y nunca tiene que esperar. En los sistemas de bases de datos habituales, en los que la lectura es una operación mucho más frecuente que la escritura, esta ventaja puede tener un mayor significado práctico.

Este esquema, sin embargo, tiene dos propiedades no deseables. En primer lugar, la lectura de un elemento de datos necesita también la actualización del campo marca-temporal-L, lo que tiene como resultado dos accesos potenciales al disco en lugar de uno. En segundo lugar, los conflictos entre las transacciones se resuelven por medio de retrocesos en lugar de esperas. Esta alternativa puede ser costosa. En la Sección 15.6.2 se describe un algoritmo que evita este problema.

Este esquema de ordenación por marcas temporales multiversión no asegura la recuperabilidad ni la ausencia de cascadas. Se puede extender de la misma forma que el esquema de ordenación por marcas temporales básico, para hacerlo recuperable y sin cascadas.

15.6.2. Bloqueo de dos fases multiversión

El **protocolo de bloqueo de dos fases multiversión** intenta combinar las ventajas del control de concurrencia multiversión con las ventajas del bloqueo de dos fases. Este protocolo distingue entre **transacciones de solo lectura** y **transacciones de actualización.**

Las transacciones de actualización realizan un bloqueo de dos fases riguroso; es decir, mantienen todos los bloqueos hasta el final de la transacción. Así, se pueden secuenciar según su orden de compromiso. Cada versión de un elemento de datos tiene una única marca temporal. La marca temporal no es en este caso una marca temporal basada en un reloj real, sino que es un contador, que se denomina contador-mt, que se incrementa durante el procesamiento del compromiso.

El sistema de bases de datos asigna a las transacciones de solo lectura una marca temporal leyendo el valor actual de contador-mt antes de que comiencen su ejecución; para realizar las lecturas siguen el protocolo de ordenación por marcas temporales multiversión. Así, cuando una transacción de solo lectura T_i ejecuta leer(Q), el valor que se devuelve es el contenido de la versión con la mayor marca temporal menor o igual que MT(T_i).

Cuando una transacción de actualización lee un elemento, obtiene un bloqueo compartido sobre él y lee la versión más reciente de dicho elemento de datos. Cuando una transacción de actualización quiere escribir un elemento, obtiene primero un bloqueo exclusivo sobre el elemento, y luego crea una nueva versión del elemento de datos. La escritura se realiza sobre la nueva versión y la marca temporal de la nueva versión tiene inicialmente el valor ∞, un valor mayor que el de cualquier marca temporal posible.

Una vez que la transacción T_i ha completado sus acciones, realiza el procesamiento del compromiso como sigue: en primer lugar, T_i asigna la marca temporal de todas las versiones que ha creado al valor de contador-mt más 1; después, T_i incrementa contador-mt en 1. Solo se permite que realice el procesamiento del compromiso a una transacción de actualización a la vez.

Como resultado, las transacciones de solo lectura que comiencen después de que T_i incremente contador-mt observarán los valores que T_i ha actualizado, mientras que aquellas que comiencen antes de que T_i incremente contador-mt observarán los valores anteriores a la actualización de T_i. En ambos casos las transacciones de solo lectura no tienen que esperar nunca por los bloqueos. El bloqueo de dos fases multiversión también asegura que las planificaciones son recuperables y sin cascadas.

Las versiones se borran de forma similar a la utilizada en la ordenación por marcas temporales multiversión. Suponga que hay dos versiones, Q_k y Q_j, de un elemento de datos y que ambas versiones tienen una marca temporal menor o igual que la de la transacción de solo lectura más antigua del sistema. Entonces la versión más antigua de Q_k y Q_j no se volverá a utilizar y se puede borrar.

15.7. Aislamiento de instantáneas

El aislamiento de instantáneas es un tipo particular de esquema de control de concurrencia que ha ganado una gran aceptación en los sistemas comerciales y de fuente abierta, incluyendo Oracle, PostgreSQL y SQLServer. Se realizó una introducción al aislamiento de instantáneas en la Sección 14.9.3. En esta sección se tratará más en detalle su funcionamiento.

Conceptualmente, un aislamiento de instantánea implica dar a una transacción una «instantánea» de la base de datos en el momento en que comienza su ejecución. Entonces, trabaja sobre la instantánea en completo aislamiento de otras transacciones concurrentes. Los valores de los datos en la instantánea solo constan de los datos que escriben las transacciones comprometidas. Este aislamiento es ideal para las transacciones de solo lectura ya que nunca tendrán que esperar y nunca son abortadas por el gestor de concurrencia. Las transacciones que realizan actualizaciones en la base de datos deben, por supuesto, interaccionar con otras transacciones de actualización potencialmente conflictivas antes de que se realicen realmente las actualizaciones en la base de datos. Las actualizaciones se mantienen en el espacio de trabajo privado de la transacción hasta que esta termina correctamente con su compromiso, momento en que las actualizaciones se escriben en la base de datos. Cuando a una transacción T se le permite hacer comprometer, se debe realizar como una acción atómica la transición de T al estado de comprometido y la escritura de todas las actualizaciones de T en la base de datos, por lo que cualquier instantánea que cree otra transacción o incluye todas las actualizaciones de T o no incluye ninguna.

15.7.1. Pasos de validación para las transacciones de actualización

Hay que decidir con cuidado si se permite a una transacción de actualización realizar el compromiso. Potencialmente, dos transacciones que se ejecutan concurrentemente podrían actualizar el mismo elemento de datos. Como las dos transacciones funcionan en aislamiento usando sus propias instantáneas privadas, ninguna transacción ve las actualizaciones realizadas por la otra. Si a ambas transacciones se les permite escribir en la base de datos, la primera actualización será sobrescrita por la segunda. El resultado es una **actualización perdida.** Claramente hay que prevenirlo. Existen dos variantes del aislamiento de instantáneas, en las que ambas previenen las actualizaciones perdidas. Se denominan *primer compromiso gana* y *primera actualización gana*. Ambos enfoques se basan en comprobar la transacción contra las transacciones concurrentes. Se dice que una transacción es **concurrente con** T si estaba activa o parcialmente comprometida en algún momento desde el inicio de T, incluyendo el momento en que se inició la comprobación.

Bajo el esquema **primer compromiso gana,** cuando una transacción T entra en el estado parcialmente comprometido, en una acción atómica se toman las siguientes medidas:

- Se realiza una comprobación para ver si alguna transacción que fuera concurrente con T ha escrito alguna actualización en la base de datos de algún elemento de datos que T pretenda escribir.
- Si se encuentra dicha transacción, T aborta.
- Si no se encuentra, entonces T compromete y sus actualizaciones se escriben en la base de datos.

Esta aproximación se denomina «primer compromiso gana» porque si las transacciones entran en conflicto, la primera que se compruebe utilizando la regla anterior consigue escribir sus actualizaciones, mientras que a las siguientes se les fuerza a abortar. Los detalles sobre cómo implementar las comprobaciones anteriores se tratan en el Ejercicio práctico 15.19.

Bajo el esquema **primera actualización gana,** el sistema usa un mecanismo de bloqueo que solo se aplica a las actualizaciones (las lecturas no se ven afectadas, ya que no obtienen bloqueos). Cuando una transacción T_i intenta actualizar un elemento de datos, solicita un *bloqueo de escritura* sobre dicho elemento. Si el bloqueo no lo tiene una transacción concurrente, se llevan a cabo los siguientes pasos tras conseguir el bloqueo:

- Si el elemento lo ha actualizado cualquier transacción concurrente, T_i aborta.
- En otro caso, T_i puede continuar con su ejecución, incluyendo posiblemente el compromiso.

Sin embargo, si alguna otra transacción concurrente T_j ya ha conseguido el bloqueo sobre el elemento de dato, entonces T_i no puede continuar y se siguen las reglas:

- T_i espera hasta que T_j aborta o compromete.
  - Si T_j aborta, entonces se libera el bloqueo y T_i puede obtenerlo. Tras conseguir el bloqueo, se realiza la comprobación de actualización por una transacción concurrente como se ha indicado anteriormente: T_i aborta si una transacción concurrente ha actualizado el elemento de dato, y continúa con su ejecución en caso contrario.
  - Si T_j compromete, entonces T_i debe abortar.

Los bloqueos se liberan cuando las transacciones comprometen o abortan.

Esta aproximación se llama «primera actualización gana» porque si las transacciones entran en conflicto, la primera que obtenga el bloqueo es a la que se permite el compromiso y realiza la actualización. Aquellas que intenten realizar la actualización después abortan, salvo que la primera que actualiza se aborte por alguna otra razón. (Como alternativa a esperar a ver si la primera que actualiza T_j aborta, una planificación subsiguiente T_i aborta en cuanto descubre que el bloqueo de escritura que solicita lo tiene T_j).

15.7.2. Sobre la secuencialidad

El aislamiento de instantáneas resulta atractivo en la práctica porque la sobrecarga es baja y no ha lugar a abortar a no ser que dos transacciones concurrentes actualicen el mismo dato.

Sin embargo, existe un importante problema con el aislamiento de instantáneas tal como se ha presentado, así como según se implementa en la práctica: el *aislamiento de instantáneas* **no** *asegura la secuencialidad*. Es cierto hasta en Oracle, que usa el aislamiento de instantáneas ¡para la implementación del nivel de aislamiento secuenciable! A continuación se dan algunos ejemplos de ejecuciones no secuenciables bajo aislamiento de instantáneas y se muestra cómo tratarlos.

1. Suponga que existen dos transacciones concurrentes T_i y T_j y dos elementos de datos A y B. Suponga que T_i lee A y B, y después actualiza B, mientras que T_i lee A y B, y después actualiza A. Por simplicidad, se supone que no existen otras transacciones concurrentes. Como T_i y T_j son concurrentes, ninguna de las transacciones ve las modificaciones del otro. Pero como actualizan diferentes elementos de datos, se permite a ambas realizar el compromiso independientemente de si el sistema usa la directiva primera-actualización-gana o primer-compromiso-gana.

   Sin embargo, el grafo de precedencia tiene un ciclo. Existe una arista en el grafo de precedencia de T_i a T_j porque T_i lee el valor de la A que existió antes de que T_j escribiese la A. También existe una arista en el grafo de precedencia de T_j a T_i ya que T_j lee el valor de B que existió antes de que T_i escribiese B. Como existe un ciclo en el grafo de precedencia, el resultado es una planificación no secuenciable.

   Esta situación, en la que cada elemento del par de transacciones ha leído el dato que ha escrito la otra pero no hay ningún dato que escriban ambas transacciones, se denomina **atasco de escritura.** Como ejemplo concreto, considere el escenario de un banco. Suponga que el banco fuerza la restricción de integridad de forma que la suma de todos los saldos en las cuentas corrientes y cuentas de ahorro de un cliente no debe ser negativa. Suponga que los saldos de las cuentas corrientes y de ahorro de un cliente son de 100 € y 200 €, respectivamente. Suponga que la transacción T_{36} extrae 200 € de la cuenta corriente, después de verificar la restricción de integridad leyendo ambos saldos. Suponga que concurrentemente la transacción T_{37} extrae 200 € de la cuenta de ahorros, de nuevo tras comprobar la restricción de integridad. Como las dos transacciones verifican la restricción de integridad en su propia instantánea, si se ejecutan concurrentemente, cada una comprobará que la suma de los saldos tras extraer el dinero será de 100 € y, por tanto, esa retirada no viola la restricción. Como las dos transacciones actualizan distintos elementos de datos, no tienen ningún conflicto de actualización y bajo el aislamiento de instantánea ambas pueden comprometerse.

   Desafortunadamente, en el estado final, después de que T_{36} y T_{37} hayan comprometido, la suma de los saldos es de 100 €, violando la restricción de integridad. Esta violación no habría ocurrido nunca en una ejecución secuencial de T_{36} y T_{37}.

   Hay que indicar que las restricciones de integridad que fuerza la base de datos, como las restricciones de clave primaria y de clave externa, no se pueden comprobar en la instantánea; en caso contrario, podría ocurrir que dos transacciones concurrentes insertasen dos tuplas con el mismo valor de clave primaria, o que una transacción insertase un valor de clave externa que concurrentemente se borra de su tabla de referencia. Al contrario, el sistema de base de datos debe comprobar estas restricciones en el estado actual de la base de datos como parte de la validación en el momento del compromiso.

2. Para el siguiente ejemplo, considere dos transacciones de actualización concurrentes que no presentan ningún problema entre ellas en cuanto a la secuencialidad, menos con una transacción de solo lectura que aparece en el momento preciso como para originar un problema.

   Suponga que tenemos dos transacciones concurrentes T_i y T_j y dos elementos de datos A y B. Suponga que T_i lee B y después actualiza B, mientras que T_j lee A y B y después actualiza A. Al ejecutar estas dos transacciones de forma concurrente no aparece ningún problema. Como T_i solo accede al dato B, no existen conflictos sobre el dato A y, por tanto, no existen ciclos en el grafo de precedencia. La única arista del grafo va de T_j a T_i, ya que T_j lee el valor de B que existía antes de que T_i escribiese B.

   Sin embargo, suponga que T_i compromete mientras que T_j todavía está activa. Suponga que tras el compromiso de T_i, pero antes del de T_j, entra una nueva transacción T_k de solo lectura que lee tanto A como B. Su instantánea incluye la actualización de T_i, ya que T_i ya comprometió. Sin embargo, como T_j no lo ha hecho todavía, su actualización aún no se ha escrito en la base de datos y no está en la instantánea que tiene T_k.

   Considere las aristas que se añaden al grafo de precedencia debido a T_k. Aparece una arista de T_i a T_k, porque T_i escribe el valor de B que existía antes de que T_k leyera B. Aparece, también, una arista de T_k a T_j, ya que T_k lee el valor de A que existía antes de que T_j escribiese A. Esto genera un ciclo en el grafo de precedencia, lo que demuestra que la planificación resultante no es secuenciable.

Las anomalías anteriores no son tan problemáticas como aparentan. Recuerde que la razón de la secuencialidad es asegurar que, independientemente de la ejecución concurrente de transacciones, se conserva la consistencia. Como el objetivo es la consistencia, es aceptable que existan ejecuciones no secuenciables si se asegura que las ejecuciones no secuenciables que pudiesen producirse no generar inconsistencias. El segundo de los ejemplos anteriores solo es un problema si para la aplicación que realiza la transacción de solo lectura (T_k) es importante que no obtenga las actualizaciones de A y B en orden. En este ejemplo no se especifican las restricciones de consistencia de la base de datos que cada transacción espera que se cumplan. Si se trata de una base de datos financiera, pudiera ser un problema que T_k lea las actualizaciones en orden distinto. Por otra parte, si A y B son matriculaciones en dos secciones de la misma asignatura, T_k no puede pedir una secuencialidad perfecta y de la aplicación se puede deducir que las tasas de actualización son suficientemente bajas como para que cualquier inexactitud que pueda leer T_k no sea significativa.

El hecho de que la base de datos deba comprobar las restricciones de integridad en el momento de compromiso y no en la instantánea, también ayuda a evitar inconsistencias en algunas situaciones. Algunas aplicaciones financieras generan secuencias de números consecutivos, por ejemplo para numerar facturas, empezando por el número más alto hasta ese momento y añadiendo 1 para obtener el siguiente valor. Si dos de estas transacciones se ejecutan concurrentemente, ambas verían el mismo conjunto de facturas en su instantánea y crearían una nueva factura con el mismo número. Ambas pasarían la comprobación de validación del aislamiento de instantánea, ya que no actualizan ninguna tupla común. Sin embargo, la ejecución no es secuenciable; el estado de la base de datos no se puede obtener de ninguna ejecución secuencial de las dos transacciones. La creación de dos facturas con el mismo número puede tener serias implicaciones legales.

El problema anterior es un ejemplo del fenómeno fantasma, ya que la inserción que realiza cada una de las transacciones entra en conflicto con la lectura que realiza la otra para encontrar la factura con el número más alto, pero el conflicto no se detecta debido al aislamiento de instantáneas.[1]

Por suerte, en la mayoría de estas aplicaciones el número de factura se declara como clave primaria, y el sistema de bases de datos detectaría la violación de clave primaria fuera de la instantánea y forzaría la vuelta atrás de las dos transacciones.[2]

El desarrollador de aplicaciones se puede proteger contra ciertas anomalías del aislamiento de instantánea añadiendo la cláusula **for update** a la consulta de SQL, como se muestra a continuación:

```
select*
from profesor
where ID = 22222
for update;
```

Al añadir la cláusula **for update,** el sistema trata los datos que se leen como si se hubiesen actualizado con el objetivo de realizar un control de concurrencia. En el primer ejemplo, en el atasco de escritura, si se añade la cláusula **for update** a la cláusula de la consulta de selección que lee los saldos de las cuentas, solo se permitiría que comprometiese una de las dos transacciones concurrentes, ya que parecería que ambas transacciones están actualizando tanto el saldo de la cuenta corriente como el de la de ahorro.

1 La norma de SQL utiliza el término problema fantasma para referirse a lecturas no repetibles de predicados, lo que lleva a algunos a indicar que el aislamiento de instantánea evita el problema fantasma; sin embargo, esto no es válido bajo la definición de conflicto fantasma.

2 El problema de los números de facturas duplicados ocurrió en varias ocasiones en una aplicación financiera en I.I.T. Bombay, donde (por razones muy complejas que no se van a tratar aquí) los números de factura no eran clave primaria, y lo detectaron los auditores de cuentas.

En el segundo ejemplo de ejecución no secuenciable, si el autor de la transacción T_k quisiera evitar esta anomalía, la cláusula **for update** se añadiría al **select** de la consulta aunque, de hecho, no exista ninguna actualización. En nuestro ejemplo, si T_k usase **select for update,** se trataría como si hubiese una actualización de A y de B cuando las leyese. El resultado sería que tanto T_k como T_j podrían abortarse y reintentarlo más tarde como una nueva transacción. Esto conduciría a una ejecución secuenciable. En este ejemplo, las consultas de las otras dos transacciones no necesitan la cláusula **for update;** un uso innecesario de la cláusula **for update** puede generar una reducción significativa de la concurrencia.

Existen métodos formales (consulte las notas bibliográficas) para determinar si un conjunto dado de transacciones corre el riesgo de una ejecución no secuenciable bajo un aislamiento de instantánea, y decidir en qué partes conflictivas hacer modificaciones (usando por ejemplo la cláusula **for update**) para asegurar la secuencialidad. Por supuesto, tales métodos solo pueden aplicarse si se conoce con antelación qué transacciones se van a ejecutar. En algunas aplicaciones, todas las transacciones pertenecen a un determinado conjunto de transacciones, lo que hace posible el análisis. Sin embargo, si la aplicación permite el uso sin restricciones de transacciones, este análisis no es posible.

De los tres sistemas más utilizados que disponen de aislamiento de instantánea, SQL Server ofrece la opción de un nivel de aislamiento *secuenciable* que asegura verdaderamente la secuencialidad, junto con un nivel de aislamiento de *instantánea* que proporciona las ventajas del rendimiento del aislamiento de instantánea (junto con el potencial de las anomalías tratadas anteriormente). En Oracle y PostgreSQL el nivel de aislamiento *secuenciable* solo ofrece aislamiento de instantánea.

15.8. Operaciones para insertar y borrar, y lectura de predicados

Hasta ahora se ha centrado la atención en las operaciones leer y escribir. Esta ligadura limita las transacciones a los elementos de datos que ya están en la base de datos. Algunas transacciones necesitan no solo acceder a los elementos de datos existentes, sino también poder crear nuevos elementos de datos. Otras necesitan tener la posibilidad de borrar elementos de datos. Para examinar la forma en que estas transacciones afectan al control de concurrencia se introducen las siguientes operaciones adicionales:

- **borrar***(Q)*: borra de la base de datos el elemento de datos Q.
- **insertar***(Q)*: inserta en la base de datos el nuevo elemento de datos Q y le asigna un valor inicial.

Si la transacción T_i intenta ejecutar una operación leer*(Q)* después de haberse borrado Q, se produce un error lógico en T_i. Igualmente, si la transacción T_i intenta ejecutar una operación leer*(Q)* antes de que Q se haya insertado, se produce un error lógico en T_i. También es un error lógico intentar borrar un dato inexistente.

15.8.1. Borrado

Para comprender la manera en que puede afectar la presencia de las instrucciones **borrar** al control de concurrencia, se debe decidir cuándo una instrucción **borrar** entra en conflicto con otra instrucción. Sean I_i e I_j instrucciones de T_i y T_j, respectivamente, que son consecutivas en la planificación S. Sea I_i = **borrar**(Q). Se consideran distintas instrucciones I_j:

- I_j = leer*(Q)*. I_i e I_j están en conflicto. Si I_i está antes de I_j, T_j tendrá un error lógico. Si I_j está antes de I_i, T_j puede ejecutar con éxito su operación leer.

- I_j = escribir(Q). I_i e I_j están en conflicto. Si I_i está antes de I_j, T_j tendrá un error lógico. Si I_j está antes de I_i, T_j puede ejecutar con éxito su operación escribir.
- I_j = borrar(Q). I_i e I_j están en conflicto. Si I_i está antes de I_j, T_j tendrá un error lógico. Si I_j está antes de I_i, T_i tendrá un error lógico.
- I_j = insertar(Q). I_i e I_j están en conflicto. Suponga que no existe el elemento de datos Q antes de la ejecución de I_i e I_j. Entonces, si I_i está antes de I_j, hay un error lógico en T_i. Si I_j está antes de I_i, entonces no hay ningún error lógico. Igualmente, si Q existía antes de la ejecución de I_i e I_j, entonces hay un error lógico si I_j está antes de I_i, pero no en otro caso.

Se puede concluir lo siguiente:

- En el protocolo de bloqueo de dos fases, se necesita un bloqueo exclusivo en un elemento de datos antes de que se borre dicho elemento.
- En el protocolo de ordenación por marcas temporales, se debe hacer una prueba similar a la que se hacía con escribir. Suponga que la transacción T_i ejecuta borrar(Q).
  - Si MT(T_i) < marca-temporal-L(Q), entonces el valor de Q que va a borrar T_i lo ha leído ya otra transacción T_j con MT(T_j) > MT(T_i). Por tanto, se rechaza la operación borrar y T_i retrocede.
  - Si MT(T_i) < marca-temporal-E(Q), entonces la transacción T_j con MT(T_j) > MT(T_i) ha escrito Q. Por tanto, se rechaza esta operación borrar y T_i retrocede.
  - En otro caso se ejecuta la operación borrar.

15.8.2. Inserción

Ya se ha visto que una operación insertar(Q) entra en conflicto con una operación borrar(Q). De forma similar, insertar(Q) entra en conflicto con las operaciones leer(Q) y escribir(Q). No se pueden realizar operaciones leer o escribir sobre un elemento de datos hasta que este último exista.

Puesto que insertar(Q) asigna un valor al elemento de datos Q, desde el punto de vista del control de concurrencia, insertar se trata de forma similar a escribir.

- En el protocolo de bloqueo de dos fases, si T_i realiza un operación insertar(Q), se da a T_i un bloqueo exclusivo sobre el nuevo elemento de datos Q creado.
- En el protocolo de ordenación por marcas temporales, si T_i realiza una operación insertar(Q), se fijan los valores marca-temporal-L(Q) y marca-temporal-E(Q) a MT(T_i).

15.8.3. Lectura de predicados y el fenómeno fantasma

Considere una transacción T_{30} que ejecuta la siguiente consulta SQL a la base de datos de la universidad:

```
select count(*)
from profesor
where nombre_dept = «Física»;
```

La transacción T_{30} necesita acceder a todas las tuplas de la relación *profesor* que pertenezcan al departamento de Física.

Sea T_{31} una transacción que ejecuta la siguiente inserción en SQL:

```
insert into profesor
  values (11111, «Feynman», «Física», 94000);
```

Sea S una planificación que involucra a T_{30} y T_{31}. Se espera que haya un conflicto potencial por las siguientes razones:

- Si T_{30} utiliza la tupla que acaba de insertar T_{31} al calcular **count**(*), entonces T_{30} lee el valor que ha escrito T_{31}. Así, en una planificación secuencial equivalente a S, T_{31} debe ir antes de T_{30}.
- Si T_{30} no utiliza la tupla que acaba de insertar T_{31} al calcular **count**(*), entonces en una planificación secuencial equivalente a S, T_{30} debe ir antes de T_{31}.

El segundo caso de estos dos es curioso. T_{30} y T_{31} no acceden a ninguna tupla común, ¡y sin embargo entran en conflicto! En efecto, T_{30} y T_{31} están en conflicto en una tupla fantasma. Si se realiza el control de concurrencia con granularidad de tupla, dicho conflicto no se detecta. Como resultado, el sistema podría no impedir una planificación no secuenciable. Este problema recibe el nombre de **fenómeno fantasma.**

Además del fenómeno fantasma hay que tratar la situación que se vio en la Sección 14.10, en la que una transacción utilizaba un índice para encontrar solo las tuplas con *nombre_dept* = «Física» y, por tanto, no leía tuplas con otros nombres de departamento. Si otra transacción actualizaba una de dichas tuplas, modificando el nombre del departamento a Física, se producía un problema equivalente al del problema fantasma. Ambos problemas se producen en las lecturas de predicados y tienen una solución común.

Para evitar estos problemas, se permite que la transacción T_{30} impida a otras transacciones crear nuevas tuplas en la relación *profesor* con *nombre_dept* = «Física», y actualizar el nombre de departamento de una tupla *profesor* existente a Física.

Para encontrar todas las tuplas de *profesor* con *nombre_dept* = «Física», T_{30} debe buscar o bien en toda la relación *profesor*, o al menos en un índice de la relación. Hasta ahora se ha supuesto implícitamente que los únicos elementos de datos a los que accede una transacción son tuplas. Sin embargo, T_{30} es un ejemplo de transacción que lee información sobre qué tuplas pertenecen a una relación, y T_{31} es un ejemplo de transacción que actualiza dicha información.

Claramente no es suficiente bloquear las tuplas a las que se accede; también se necesita bloquear la información utilizada para encontrar a qué tuplas accede la transacción.

El bloqueo de la información utilizada para encontrar las tuplas se puede implementar asociando un elemento de datos con la propia relación; el elemento de datos representa la información utilizada para encontrar las tuplas en la relación. Las transacciones como T_{30}, que lean la información sobre qué tuplas pertenecen a la relación, tendrían que bloquear el elemento de datos correspondientes a la relación en modo compartido. Las transacciones como T_{31}, que actualicen la información sobre qué tuplas pertenecen a la relación, tendrían que bloquear el elemento de datos en modo exclusivo. De este modo, T_{30} y T_{31} tendrían un conflicto en un elemento de datos real, en lugar de tenerlo en uno fantasma. De forma similar, las transacciones que usan un índice para recuperar las tuplas deben bloquear el propio índice.

No se debe confundir el bloqueo de una relación completa, como en el bloqueo de granularidad múltiple, con el bloqueo del elemento de datos correspondiente a la relación. Al bloquear el elemento de datos, la transacción solo evita que otras transacciones actualicen la información acerca de qué tuplas pertenecen a la relación. Sigue siendo necesario un bloqueo de tuplas. Una transacción que acceda directamente a una tupla puede obtener un bloqueo sobre las tuplas incluso si otra transacción posee un bloqueo exclusivo sobre el elemento de datos correspondiente a la propia relación.

El mayor inconveniente de bloquear un elemento de datos correspondiente a la relación, o el bloqueo de un índice completo, es el bajo grado de concurrencia; se impide que dos transacciones que inserten distintas tuplas en la relación se ejecuten concurrentemente.

Una solución mejor es la técnica de **bloqueo del índice,** que evita bloquear el índice completo. Toda transacción que inserte una tupla en una relación debe insertar información en cada uno de los índices que se mantengan en la relación. El fenómeno fantasma se elimina al imponer un protocolo para los índices. Para simplificar, solo se van a considerar los índices del árbol B^+.

Como se vio en el Capítulo 11, todo valor de la clave de búsqueda se asocia a un nodo hoja índice. Una consulta usará normalmente uno o más índices para acceder a la relación. Una inserción debe insertar una nueva tupla en todos los índices de la relación. En el ejemplo, se asume que hay un índice para *profesor* en *nombre_dept*. Entonces, T_{31} debe modificar la hoja que contiene la clave «Física». Si T_{30} lee el mismo nodo hoja para localizar todas las tuplas que pertenecen al departamento Física, entonces T_{30} y T_{31} tienen un conflicto en dicho nodo hoja.

El **protocolo de bloqueo de índices** se aprovecha de la disponibilidad de índices en una relación convirtiendo las apariciones del fenómeno fantasma en conflictos de los bloqueos sobre los nodos hoja índice. El protocolo opera de la siguiente manera:

- Toda relación debe tener al menos un índice.
- Una transacción T_i puede acceder a las tuplas de una relación únicamente después de haberlas encontrado primero a través de uno o más índices de la relación. Para el objetivo del protocolo de bloqueo de índices, una exploración de la relación se trata como una exploración por todas las hojas de uno de los índices.
- Una transacción T_i que realiza una búsqueda (ya sea una búsqueda de rango o una búsqueda concreta) debe bloquear en modo compartido todos los nodos hoja índice a los que accede.
- Una transacción T_i no puede insertar, borrar ni actualizar una tupla t_i en una relación r sin actualizar todos los índices de r. La transacción debe obtener bloqueos en modo exclusivo sobre todos los nodos hoja índice que están afectados por la inserción, el borrado o la actualización. Para la inserción y el borrado, los nodos hoja afectados son aquellos que contienen (después de la inserción) o han contenido (antes del borrado) el valor de la clave de búsqueda en la tupla. Para las actualizaciones, los nodos hoja afectados son los que (antes de la modificación) contenían el valor antiguo de la clave de búsqueda y los nodos que (después de la modificación) contienen el nuevo valor de la clave de búsqueda.
- Los bloqueos se obtienen sobre las tuplas de la forma usual.
- Hay que cumplir las reglas del protocolo de bloqueo de dos fases.

Tenga en cuenta que el protocolo de bloqueo de índices no trata el control de concurrencia sobre los nodos internos del índice; las técnicas sobre el control de concurrencia en índices, que minimizan los conflictos de bloqueo, se tratan en la Sección 15.10.

El bloqueo de un nodo hoja índice evita cualquier actualización del nodo, incluso si la actualización no entra en conflicto con el predicado. Una variante, denominada bloqueo por valor de clave, que minimiza estos conflictos de bloqueo falsos, se presenta en la Sección 15.10 como parte del control de concurrencia de índices.

Como se ha indicado en la Sección 14.10, parecería como si la existencia de un conflicto entre las transacciones dependiese de una decisión del procesamiento de consultas de bajo nivel por parte del sistema que no está relacionado con el nivel de usuario sobre el significado de las dos transacciones. Un enfoque alternativo al control de concurrencia adquiere los bloqueos compartidos sobre los predicados de una consulta, como en el predicado «*sueldo* > 90000» en la relación *profesor*. Después hay que comprobar las inserciones y borrados de la relación para ver si satisfacen el predicado; si es así, existe un conflicto de bloqueo, lo que fuerza a la inserción o borrado a esperar hasta que el bloqueo del predicado se libera. Para las actualizaciones, se debe comprobar para el predicado tanto el valor inicial como final. Estos conflictos de inserción, borrado y actualización afectan al conjunto de tuplas seleccionadas por el predicado y no se pueden ejecutar concurrentemente con la consulta que adquirió el bloqueo de predicado (compartido). Este protocolo se denomina **bloqueo de predicado;**[3] el bloqueo de predicado no se usa en la práctica, pues es más costoso de implementar que el protocolo de bloqueo de índices, y no aporta ventajas significativas.

Se pueden usar variantes de la técnica de bloqueo de predicados para eliminar el fenómeno fantasma bajo el protocolo de control de concurrencia que se presenta en este capítulo. Sin embargo, muchos sistemas de bases de datos, como PostgreSQL (en la versión 8.1) y (hasta donde conocemos) Oracle (en la versión 10g) no implementan el bloqueo de índices ni el bloqueo de predicados, y son vulnerables a la no secuencialidad debido al problema fantasma incluso si el nivel de aislamiento se establece en **secuenciable.**

15.9. Niveles débiles de consistencia en la práctica

En la Sección 14.5 se estudiaron los niveles de aislamiento que especifica la norma de SQL: secuenciable, lectura repetible, lectura comprometida y lectura no comprometida. En esta sección, primero se describe cierta terminología antigua relacionada con los niveles de consistencia más débiles que los de secuencialidad y se relacionan con los niveles de la norma SQL. Después se tratan los temas del control de consistencia para las transacciones que implican interacción con el usuario, un tema que se trató brevemente con anterioridad en la Sección 14.8.

15.9.1. Consistencia de grado dos

El objetivo de la **consistencia de grado dos** es evitar abortar en cascada sin asegurar necesariamente la secuencialidad. El protocolo de bloqueo para la consistencia de grado dos utiliza los mismos dos modos de bloqueo que se utilizan para el protocolo de bloqueo de dos fases: compartido (C) y exclusivo (X). Las transacciones deben mantener el modo de bloqueo adecuado cuando acceden a un elemento de datos, pero no deben cumplir el comportamiento de dos fases.

A diferencia de la situación en los bloqueos de dos fases, los bloqueos C pueden liberarse en cualquier momento y también se pueden adquirir bloqueos en cualquier momento. Sin embargo, los bloqueos exclusivos no se pueden liberar hasta que la transacción se comprometa o se aborte. La secuencialidad no queda asegurada por este protocolo. En realidad, una transacción puede leer dos veces el mismo elemento de datos y obtener resultados diferentes. En la Figura 15.20, T_{32} lee el valor de Q antes y después de que T_{33} escriba su valor.

T_{32}	T_{33}
bloquear-c*(Q)*	
leer*(Q)*	
desbloquear*(Q)*	
	bloquear-x*(Q)*
	leer*(Q)*
	escribir*(Q)*
	desbloquear*(Q)*
bloquear-c*(Q)*	
leer*(Q)*	
desbloquear*(Q)*	

Figura 15.20. Planificación no secuenciable con consistencia de grado dos.

3 El término *bloqueo de predicado* se utilizó para una versión del protocolo que usaba bloqueos exclusivos y compartidos sobre los predicados, y era, por tanto, más complicado. La versión que se presenta con solo bloqueos compartidos sobre predicados también se denomina **bloqueo de precisión.**

Claramente, la lectura no es repetible, pero como se mantienen bloqueos exclusivos hasta que la transacción queda comprometida, ninguna transacción puede leer un valor no comprometido. Por tanto, la consistencia de grado dos es una implementación particular del nivel de aislamiento de lectura comprometida.

15.9.2. Estabilidad del cursor

La **estabilidad del cursor** es una forma de consistencia de grado dos diseñada para programas que iteran sobre las tuplas de una relación utilizando cursores. En vez de bloquear toda la relación, la estabilidad del cursor asegura que:

- La tupla que está procesando la iteración esté bloqueada en modo compartido.
- Todas las tuplas modificadas estén bloqueadas en modo exclusivo hasta que se comprometa la transacción.

Estas reglas aseguran que se obtiene consistencia de grado dos. No se requiere bloqueo de dos fases. No se garantiza la secuencialidad. La estabilidad del cursor se utiliza en la práctica sobre relaciones a las que muchas veces se accede como una forma de incrementar la concurrencia y mejorar el rendimiento del sistema. Las aplicaciones que utilizan estabilidad del cursor deben ser desarrolladas de forma que aseguren la consistencia de la base de datos a pesar de la posibilidad de planificaciones no secuenciables. Por tanto, el uso de estabilidad del cursor está limitado a determinadas situaciones con restricciones simples de consistencia.

15.9.3. Control de concurrencia en interacciones de usuario

Los protocolos de control de concurrencia normalmente consideran que las transacciones no implican interacción con el usuario. Considere el ejemplo de selección de asiento de una aerolínea de la Sección 14.8, que implica interacción con el usuario. Suponga que se tienen en cuenta todos los pasos desde que se muestra la disponibilidad de asientos al usuario hasta que se confirma su selección, como una única transacción.

Si se usa el bloqueo de dos fases, se debería bloquear el conjunto completo de asientos de un vuelo en modo compartido hasta que el usuario haya completado la selección de asientos, y ninguna otra transacción sería capaz de actualizar la información de asignación de asientos durante este periodo. Claramente, este bloqueo resulta una muy mala idea, ya que el usuario puede tardar mucho tiempo en realizar la selección, o incluso abandonar la transacción sin cancelarla. En su lugar se usarían protocolos de marcas de tiempo o de validación, que evitan el problema del bloqueo, pero ambos abortan la transacción de un usuario A si otro usuario B ha actualizado la información de asignación de asientos, incluso aunque la selección de B no entre en conflicto con la selección de asientos realizada por A. El aislamiento de instantánea es una buena elección para esta situación, ya que no abortaría la transacción del usuario A mientras B no seleccione el mismo asiento que A.

Sin embargo, el aislamiento de instantánea requiere que la base de datos recuerde la información sobre qué actualizaciones realiza cada transacción, incluso después de estar comprometida, mientras haya alguna otra transacción concurrente activa, lo que puede ser un problema para transacciones que tarden mucho tiempo.

Otra opción es dividir una transacción que implica interacción con el usuario en dos o más transacciones, de forma que ninguna abarque una interacción de usuario. Si la selección de asientos se divide, la primera transacción podría leer la disponibilidad de asientos mientras que una segunda transacción completaría la asignación de los asientos seleccionados. Si la segunda transacción se escribe sin mucho cuidado, podría asignar los asientos seleccionados por el usuario sin comprobar si el asiento se ha asignado durante ese tiempo a otro usuario, lo que provoca un problema de actualización perdida. Para evitar este problema, como ya se describió en la Sección 14.8, la segunda transacción solo debería asignar el asiento si no se ha asignado ya a otro usuario.

La idea anterior se ha generalizado en un esquema de control de concurrencia alternativo que usa números de versión que se guardan en las tuplas para evitar actualizaciones perdidas. Se modifica el esquema de cada relación añadiéndole un atributo adicional *número de versión* que se inicia en 0 cuando se crea la tupla. Cuando una transacción lee (por primera vez) una tupla que intenta actualizar, se guarda el número de versión de dicha tupla. La lectura se realiza como una transacción independiente y, por tanto, cualquier bloqueo que pueda obtener se libera inmediatamente. Las actualizaciones se realizan localmente y se copian a la base de datos como parte del proceso de compromiso de acuerdo con los siguientes pasos que se ejecutan de forma atómica (es decir, como parte de una única transacción a la base de datos):

- Para cada tupla actualizada, la transacción comprueba que el número de versión actual es el mismo que cuando lo leyó la transacción.
  1. Si el número de versión coincide, se realiza la actualización de la tupla en la base de datos y se incrementa el número de versión en 1.
  2. Si el número de versión no coincide, la transacción aborta, retrocediendo todas las actualizaciones realizadas.

Si la comprobación del número de versión es correcta para todas las tuplas actualizadas, la transacción pasa a comprometida. Es importante indicar que se puede utilizar una marca de tiempo en lugar de un número de versión sin que cambie el esquema en ningún aspecto.

Observe la similitud entre el esquema anterior y el aislamiento de instantánea. La comprobación de número de versión implementa la regla primer-compromiso-gana que se utiliza en el aislamiento de instantánea y se puede usar incluso si la transacción lleva activa un gran periodo de tiempo. Sin embargo, al contrario que en el aislamiento de instantánea, la lectura que realiza la transacción puede que no se corresponda con la instantánea de la base de datos; y al contrario que el protocolo basado en validación, las lecturas que realiza la transacción no se validan.

Al esquema anterior se le denomina **control de concurrencia optimista sin validación de lectura.** El control de concurrencia optimista sin validación de lectura proporciona un nivel de secuencialidad débil y no asegura la secuencialidad. Una variante de este esquema usa números de versión para validar las lecturas al tiempo que se comprometen, además de validar las escrituras para asegurar que las tuplas que leyó la transacción no se han actualizado después de la lectura inicial; este esquema es equivalente al control de concurrencia optimista sin validación de lectura que se ha visto anteriormente.

El esquema anterior se ha usado ampliamente por desarrolladores para manejar las transacciones que implican interacción con el usuario. Una característica muy atractiva del esquema es que se puede implementar fácilmente sobre el sistema de base de datos. Los pasos de validación y actualización que se realizan formando parte del proceso de compromiso, se ejecutan como una única transacción con la base de datos, usando el esquema de control de concurrencia de la base de datos para asegurar la atomicidad del proceso de compromiso. El esquema anterior también ha sido utilizado por el sistema de asociación relacional-objetos Hibernate (Sección 9.4.2) y otros sistemas de asociación relacional-objeto, donde se alude a él como control de concurrencia optimista (incluso aunque por defecto no se validen las lecturas). Las transacciones que implican interacción con

el usuario se denominan **conversaciones** en Hibernate para diferenciarlas de la validación de las transacciones normales. Los sistemas de asociación relación-objeto también realizan caché de tuplas de la base de datos en forma de objetos en la memoria y ejecutan las transacciones sobre los objetos de la caché; las actualizaciones en los objetos se convierten en actualizaciones en la base de datos cuando la transacción pasa a comprometida. Los datos se pueden mantener en la caché durante mucho tiempo, y si las transacciones actualizan estos datos de la caché, existe un riesgo de actualización perdida. Hibernate y otros sistemas de asociación relacional-objeto, por tanto, llevan a cabo la comprobación del número de versión como parte de su proceso de compromiso. (Hibernate permite a los programadores saltarse la caché y ejecutar directamente las transacciones con la base de datos, si se desea secuencialidad).

15.10. Concurrencia en las estructuras de índices**

Es posible tratar el acceso a los índices como el de otras estructuras de base de datos y aplicar las técnicas de control de concurrencia que se han descrito anteriormente. Sin embargo, puesto que se accede frecuentemente a los índices, se pueden convertir en un punto con mucho bloqueo, lo que produce un bajo grado de concurrencia. Por suerte, no es necesario tratar a los índices como a las demás estructuras de las bases de datos. Es perfectamente aceptable que una transacción busque en un índice dos veces y se encuentre con que la estructura del índice ha cambiado entre ambas búsquedas, mientras la búsqueda devuelva el conjunto correcto de tuplas. De este modo se acepta tener un acceso no secuenciable a un índice mientras dicho índice siga siendo preciso.

A continuación se mostrarán dos técnicas para tratar los accesos concurrentes a árboles B^+. En las notas bibliográficas se hace referencia a otras técnicas para árboles B^+, así como a técnicas para otras estructuras de índice.

Las técnicas que se presentan para el control de concurrencia en los árboles B^+ se basan en el bloqueo, pero no se emplea ni el bloqueo de dos fases ni el protocolo de árbol. Los algoritmos de búsqueda, inserción y borrado son los mismos que se usaron en el Capítulo 11 con algunas pequeñas modificaciones.

La primera técnica se denomina **protocolo del cangrejo:**

- Cuando se busca un valor clave, el protocolo del cangrejo bloquea primero el nodo raíz en modo compartido. Cuando se recorre el árbol hacia abajo, adquiere un bloqueo compartido sobre el siguiente nodo hijo. Después de adquirir el bloqueo sobre el nodo hijo, libera el bloqueo sobre el nodo padre, repitiendo este proceso hasta que alcanza un nodo hoja.
- Cuando se inserta o se borra un valor clave, el protocolo del cangrejo realiza estas acciones:
  - Sigue el mismo protocolo que para la búsqueda hasta que alcanza el nodo hoja deseado. Hasta este punto, tan solo obtiene (y libera) bloqueos compartidos.
  - Bloquea el nodo hoja en modo exclusivo e inserta o borra el valor clave.
  - Si necesita dividir un nodo o fusionarlo con sus hermanos, o redistribuir los valores claves entre hermanos, el protocolo del cangrejo bloquea al padre del nodo en modo exclusivo. Después de realizar estas acciones, libera los bloqueos sobre el nodo y los hermanos.

    Si el padre requiere división, fusión o redistribución de valores clave, el protocolo mantiene el bloqueo sobre el padre; y la división, la fusión o la redistribución se siguen propagando de la misma manera. En otro caso, libera el bloqueo sobre el padre.

El protocolo obtiene su nombre de la forma en que los cangrejos avanzan, moviéndose de lado, moviendo primero las patas hacia un lado, después hacia el otro, y así alternando sucesivamente. El avance de los bloqueos mientras el protocolo baja por el árbol o sube de nuevo (en el caso de divisiones, fusiones o redistribuciones) procede de forma similar al cangrejo.

Una vez que una operación libera un bloqueo sobre un nodo, otras operaciones pueden acceder a ese nodo. Existe una posibilidad de interbloqueos entre las operaciones de búsqueda que bajan por el árbol; y las divisiones, fusiones y redistribuciones que se propagan hacia arriba por el árbol. El sistema puede manejar con facilidad tales interbloqueos reiniciando la operación de búsqueda desde la raíz, después de liberar los bloqueos mantenidos por la operación.

La segunda técnica consigue aún más concurrencia, impidiendo incluso que se mantenga un bloqueo sobre un nodo mientras se está adquiriendo el bloqueo sobre otro nodo, utilizando una versión modificada de los árboles B^+ llamados **árboles B enlazados;** los árboles B enlazados requieren que todo nodo (incluyendo los nodos internos, no solo las hojas) mantenga un puntero a su hermano derecho. Se necesita este puntero porque una búsqueda que tenga lugar mientras se divide un nodo puede que tenga que buscar no solo ese nodo sino también el hermano derecho de ese nodo (si existe alguno). Esta técnica se va a ilustrar con un ejemplo después de presentar los procedimientos modificados del **protocolo de bloqueo con árboles B enlazados.**

- **Búsqueda.** Se debe bloquear en modo compartido cada nodo del árbol B^+ antes de acceder a él. Dicho bloqueo se libera antes de que se solicite otro bloqueo sobre algún nodo del árbol B^+. Si tiene lugar una división de forma concurrente con una búsqueda, el valor de la clave de búsqueda deseado puede no aparecer dentro del rango de valores representado por un nodo al que se ha accedido en la búsqueda. En tal caso, el valor de la clave de búsqueda está en el intervalo que representa un nodo hermano, que el sistema encuentra siguiendo el puntero al hermano derecho. Sin embargo, el sistema bloquea los nodos hoja siguiendo el protocolo de bloqueo de dos fases, como se describe en la Sección 15.8.3, para evitar el fenómeno fantasma.
- **Inserción y borrado.** El sistema sigue las reglas de la búsqueda para localizar el nodo sobre el cual se va a realizar la inserción o el borrado. Se modifica el bloqueo en modo compartido sobre ese nodo a modo exclusivo y se realiza la inserción o el borrado. Se bloquean los nodos hoja afectados por la inserción o el borrado siguiendo el protocolo de bloqueo de dos fases, como se describe en la Sección 15.8.3, para evitar el fenómeno fantasma.
- **División.** Si una transacción divide un nodo se crea otro nuevo siguiendo el algoritmo de la Sección 11.3 y se convierte en el hermano derecho del nodo original. Se fijan los punteros al hermano derecho del nodo original y del nuevo nodo. Seguidamente, se libera el bloqueo en modo exclusivo sobre el nodo original (dado que es un nodo interno, los nodos hoja están bloqueados en el modo de dos fases) y se solicita un bloqueo sobre el padre para que se pueda insertar un nuevo nodo (no es necesario bloquear o desbloquear el nuevo nodo).
- **Fusión.** Si un nodo tiene muy pocos valores de clave de búsqueda después de un borrado, se debe bloquear en modo exclusivo el nodo con el que se debe fusionar. Una vez que se fusionen estos nodos, se solicita un bloqueo en modo exclusivo sobre el padre para que se pueda eliminar el nodo borrado. En ese momento se libera el bloqueo sobre los nodos fusionados. Se libera el bloqueo sobre el nodo padre, a no ser que también se tenga que fusionar.

Es importante observar que: una inserción o un borrado pueden bloquear un nodo, desbloquearlo y, posteriormente, volverlo a bloquear. Además, una búsqueda que se ejecute concurrentemente con operaciones de división o de fusión puede observar que la clave de búsqueda deseada se ha trasladado al nodo hermano derecho debido a la división o a la fusión.

Como ejemplo, considere el árbol B⁺ de la Figura 15.21. Suponga que hay dos operaciones concurrentes sobre dicho árbol B⁺:

1. Insertar «Química».
2. Buscar «Informática».

Considere que la operación inserción comienza en primer lugar. Realiza una búsqueda de «Química» y encuentra que el nodo en el cual se debe insertar está lleno. Por tanto, convierte el bloqueo compartido sobre el nodo en un bloqueo exclusivo y crea un nuevo nodo. El nodo original contiene ahora los valores de clave de búsqueda «Biología» y «Química». El nuevo nodo contiene el valor de clave de búsqueda «Informática».

Suponga ahora que se produce un cambio de contexto que le pasa el control a la operación búsqueda. Dicha operación búsqueda accede a la raíz y sigue el puntero al hijo izquierdo de la raíz. Accede entonces a ese nodo y obtiene el puntero al hijo izquierdo. El hijo izquierdo contenía originalmente los valores de clave de búsqueda «Biología» e «Informática». Puesto que la operación inserción está bloqueando actualmente en modo exclusivo dicho nodo, la operación búsqueda debe esperar. Observe que, en este punto, ¡la operación búsqueda no mantiene ningún bloqueo!

Ahora la operación inserción desbloquea el nodo hoja y vuelve a bloquear a su padre; en esta ocasión en modo exclusivo. Se completa la inserción, lo que deja al árbol B⁺ como se muestra en la Figura 15.22. Continúa la operación de búsqueda. Sin embargo, tiene un puntero a un nodo hoja incorrecto. Sigue por tanto el puntero al hermano derecho para encontrar el nodo siguiente. Si este nodo es también incorrecto, se sigue también el puntero al hermano derecho. Se puede demostrar que, si una búsqueda tiene un puntero a un nodo incorrecto, entonces, al seguir los punteros al hermano derecho, la búsqueda llega finalmente al nodo correcto.

Las operaciones de búsqueda y de inserción no pueden llevar a un interbloqueo. La fusión de nodos durante el borrado puede provocar inconsistencias, dado que una búsqueda puede tener que leer un puntero a un nodo borrado desde su padre, antes de que el nodo padre sea actualizado y, entonces, puede intentar acceder al nodo borrado. La búsqueda se tendría en este caso que reiniciar desde la raíz. Dejar los nodos sin fusionar evita tales inconsistencias. Esta solución genera nodos que contienen muy pocos valores clave de búsqueda y que violan algunas propiedades de los árboles B⁺. Sin embargo, en la mayoría de las bases de datos las inserciones son más frecuentes que los borrados, por lo que es probable que los nodos que tienen muy pocos valores clave de búsqueda ganen valores adicionales de forma relativamente rápida.

En lugar de bloquear nodos hoja índice en dos fases, algunos esquemas de control de concurrencia de índices utilizan el **bloqueo de valores clave** sobre valores clave individuales, permitiendo que se inserten o se borren otros valores clave de la misma hoja. Por lo tanto, el bloqueo de valores clave proporciona una concurrencia mejorada. Sin embargo, utilizar el bloqueo de valores clave ingenuamente podría permitir que se produjera el fenómeno fantasma; para prevenir el fenómeno fantasma se utiliza la técnica **bloqueo de la siguiente clave.** En esta técnica, cada búsqueda por índice debe bloquear no solo las claves encontradas dentro del rango (o la única clave, en caso de una búsqueda concreta), sino también el siguiente valor clave; esto es, el valor clave que es justo mayor que el último valor clave que estaba dentro del rango. Además, cada inserción no solo debe bloquear el valor que se inserta, sino también el siguiente valor clave. Así, si una transacción intenta insertar un valor que estaba dentro del rango de la búsqueda por índice de otra transacción, las dos transacciones entrarán en conflicto en el valor clave que sigue al valor clave insertado.

De forma similar, los borrados también deben bloquear el siguiente valor clave al valor que se ha borrado, para asegurarnos que se detectan los conflictos con las subsiguientes búsquedas de rango de otras consultas.

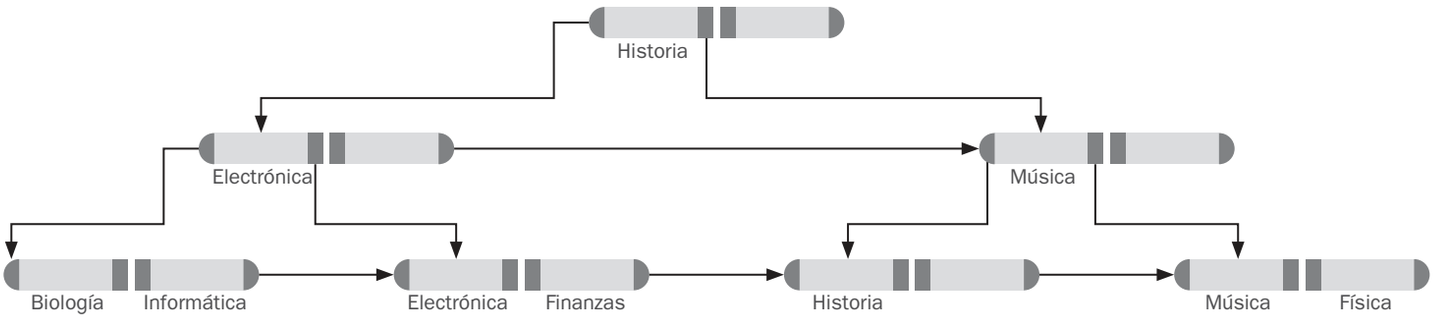


Figura 15.21. Árbol B⁺ para el archivo *departamento* con $n = 3$.

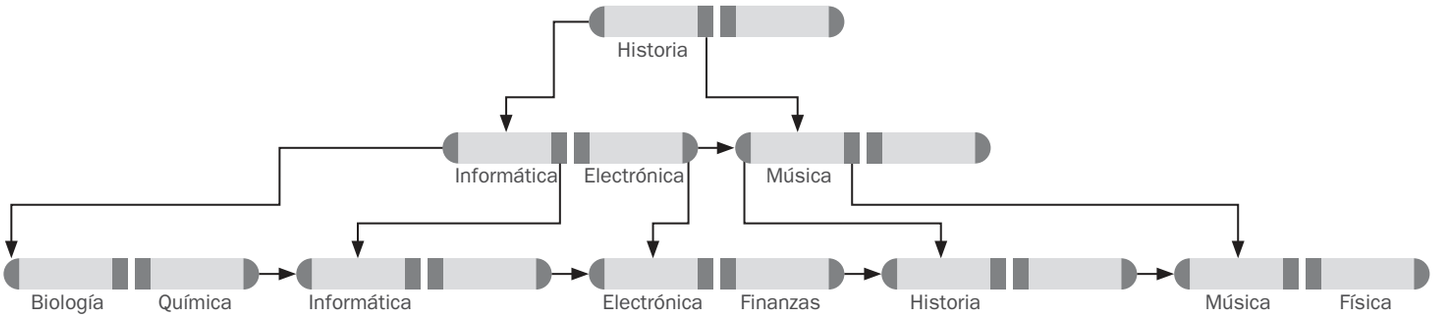


Figura 15.22. Inserción de «Química» en el árbol B⁺ de la Figura 15.21.

15.11. Resumen

- Cuando se ejecutan concurrentemente varias transacciones en la base de datos, puede dejar de conservarse la consistencia de los datos. Es necesario que el sistema controle la interacción entre las transacciones concurrentes, y dicho control se lleva a cabo mediante uno de los muchos mecanismos llamados esquemas de *control de concurrencia*.
- Se pueden usar varios esquemas de control de concurrencia para asegurar la secuencialidad. Todos estos esquemas, o bien retrasan una operación, o bien abortan la transacción que ha realizado la operación. Los más comunes son los protocolos de bloqueo, los esquemas de ordenación por marcas temporales, las técnicas de validación y los esquemas multiversión.
- Un protocolo de bloqueo es un conjunto de reglas que indican el momento en el que una transacción puede bloquear o desbloquear un elemento de datos de la base de datos.
- El protocolo de bloqueo de dos fases permite que una transacción bloquee un nuevo elemento de datos solo si todavía no ha desbloqueado ningún otro elemento de datos. Este protocolo asegura la secuencialidad pero no la ausencia de interbloqueos. A falta de información acerca de la forma en que se accede a los elementos de datos, el protocolo de bloqueo de dos fases es necesario y suficiente para asegurar la secuencialidad.
- El protocolo de bloqueo estricto de dos fases permite liberar bloqueos exclusivos solo al final de la transacción, para asegurar la recuperabilidad y la ausencia de cascadas en las planificaciones resultantes. El protocolo de bloqueo riguroso de dos fases libera todos los bloqueos solo al final de la transacción.
- Los protocolos de bloqueo basados en grafos imponen restricciones sobre el orden de acceso a los elementos, y pueden por tanto asegurar la secuencialidad sin requerir el bloqueo de dos fases, además de asegurar la ausencia de interbloqueos.
- Algunos de los protocolos de bloqueo no evitan los interbloqueos. Una forma de prevenir los interbloqueos es utilizar una ordenación de los elementos de datos, y solicitar los bloqueos en una secuencia consistente con la ordenación.
- Otra forma de prevenir los interbloqueos es utilizar la expropiación y retroceso de transacciones. Para controlar la expropiación se asigna una única marca temporal a cada transacción. El sistema utiliza estas marcas temporales para decidir si una transacción debe esperar o retroceder. Si una transacción retrocede, conserva su marca temporal anterior cuando vuelve a comenzar. El esquema herir-esperar es un esquema de expropiación.
- Si no se pueden prevenir los interbloqueos, el sistema debe ocuparse de ellos utilizando el esquema de detección y recuperación de interbloqueos. Para ello, el sistema construye un grafo de espera. Un sistema está en estado de interbloqueo si, y solo si, contiene un ciclo en el grafo de espera. Cuando el algoritmo de detección de interbloqueos determina que existe un interbloqueo, el sistema debe recuperarse del mismo. Esto se lleva a cabo retrocediendo una o más transacciones para romper el interbloqueo.
- Hay circunstancias bajo las cuales puede ser conveniente agrupar varios elementos de datos y tratarlos como un conjunto de elementos de datos por motivos relacionados con el trabajo, lo que da lugar a varios niveles de granularidad. Se permiten elementos de datos de varios tamaños y se define una jerarquía de elementos de datos en la cual los elementos más pequeños están anidados dentro de otros más grandes. Dicha jerarquía se puede representar de forma gráfica como un árbol. El orden de obtención de los bloqueos es desde la raíz hasta las hojas; se liberan desde las hojas hasta la raíz. Este protocolo asegura la secuencialidad pero no la ausencia de interbloqueos.
- El esquema de ordenación por marcas temporales asegura la secuencialidad seleccionando previamente un orden entre todo par de transacciones. Se asocia una única marca temporal fija a cada transacción del sistema. Las marcas temporales de las transacciones determinan el orden de secuencialidad. De este modo, si la marca temporal de la transacción T_i es más pequeña que la de la transacción T_j, entonces el esquema asegura que la planificación generada es equivalente a una planificación secuencial en la que la transacción T_i aparece antes de la transacción T_j. Para ello retrocede una transacción siempre que se viole dicho orden.
- Un esquema de validación es un método de control de concurrencia adecuado en aquellos casos en los que la mayoría de las transacciones son de solo lectura, y por tanto la tasa de conflictos entre dichas transacciones es baja. Se asocia una única marca temporal fija a cada transacción del sistema. El orden de secuencialidad se determina por medio de la marca temporal. En dicho esquema nunca se retrasa una transacción; sin embargo, debe pasar una comprobación de validación para poder completarse. Si no pasa la comprobación de validación, retrocede a su estado inicial.
- Un esquema de control de concurrencia multiversión se basa en crear una nueva versión de un elemento de datos cada vez que una transacción va a escribir dicho elemento. Cuando se realiza una operación de lectura, el sistema elige una de las versiones para que se lea. El esquema de control de concurrencia asegura que la versión que se va a leer se elige de forma que asegure la secuencialidad usando las marcas temporales. Una operación de lectura tiene éxito siempre.
  - En la ordenación por marcas temporales multiversión, una operación de escritura puede provocar el retroceso de una transacción.
  - En el bloqueo de dos fases multiversión, las operaciones de escritura pueden provocar una espera con bloqueo o, posiblemente, un interbloqueo.
- El aislamiento de instantáneas es un protocolo de control de concurrencia que se basa en la validación que, al contrario del bloque de dos fases multiversión, no requiere que las transacciones se declaren como de solo lectura o actualización. El aislamiento de instantáneas no garantiza la secuencialidad, pero se encuentra implantado en la mayoría de los sistemas de bases de datos.
- Solo se puede realizar una operación de **borrado** si la transacción que borra la tupla tiene un bloqueo en modo exclusivo sobre dicha tupla. A la transacción que inserta una nueva tupla se le concede un bloqueo en modo exclusivo sobre dicha tupla.
- Las inserciones pueden provocar el fenómeno fantasma, en el cual hay un conflicto entre una inserción y una consulta incluso si las dos transacciones no acceden a tuplas comunes. Este conflicto no se puede detectar si el bloqueo se ha hecho solo sobre las tuplas a las que han accedido transacciones. Es necesario bloquear los datos utilizados para encontrar las tuplas en la relación. La técnica del bloqueo del índice resuelve este problema al exigir bloqueos sobre ciertos nodos de índices. Estos bloqueos aseguran que todas las transacciones conflictivas están en conflicto por un elemento de datos real, en lugar de por uno fantasma.
- Los niveles débiles de consistencia sirven para algunas aplicaciones cuando la consistencia de los resultados de la consulta no es crítica y utilizar la secuencialidad podría dar lugar a consultas que afectaran desfavorablemente al procesamiento de transacciones. La consistencia de grado dos es uno de los niveles de consistencia débiles; la estabilidad del cursor es un caso especial de consistencia de grado dos y se utiliza ampliamente.

- El control de concurrencia es un desafío para las transacciones que implican interacción con el usuario. Las aplicaciones suelen implementar un esquema basado en la validación de las escrituras usando números de versión que se guardan en las tuplas que proporciona un nivel de secuencialidad débil y se puede implementar en el nivel de aplicación sin realizar modificaciones en la base de datos.
- Se pueden desarrollar técnicas de control de concurrencia para estructuras especiales. A menudo se aplican técnicas especiales en los árboles B^+ para permitir una mayor concurrencia. Estas técnicas permiten accesos no secuenciables al árbol B^+, pero aseguran que la estructura del árbol B^+ es correcta y que los accesos a la base de datos son secuenciables.

Términos de repaso

- Control de concurrencia.
- Tipos de bloqueo:
  - Bloqueo en modo compartido (C).
  - Bloqueo en modo exclusivo (X).
- Bloqueo:
  - Compatibilidad.
  - Solicitud.
  - Espera.
  - Concesión.
- Interbloqueo.
- Inanición.
- Protocolo de bloqueo.
- Planificación legal.
- Protocolo de bloqueo de dos fases:
  - Fase de crecimiento.
  - Fase de decrecimiento.
  - Punto de bloqueo.
  - Bloqueo estricto de dos fases.
  - Bloqueo riguroso de dos fases.
- Conversión de bloqueos:
  - Subir.
  - Bajar.
- Protocolos basados en grafos:
  - Protocolo de árbol.
  - Dependencia de compromiso.
- Tratamiento de interbloqueos:
  - Prevención.
  - Detección.
  - Recuperación.
- Prevención de interbloqueos:
  - Bloqueos ordenados.
  - Expropiación de bloqueos.
  - Esquema esperar-morir.
  - Esquema herir-esperar.
  - Esquemas basados en tiempo límite.
- Detección de interbloqueos:
  - Grafo de espera.
- Recuperación de interbloqueos:
  - Retroceso total.
  - Retroceso parcial.
- Granularidad múltiple:
  - Bloqueos explícitos.
  - Bloqueos implícitos.
  - Bloqueos intencionales.
- Modos de bloqueo intencionales:
  - Intencional-compartido (IC).
  - Intencional-exclusivo (IX).
  - Intencional-exclusivo y compartido (IXC).
- Protocolo de bloqueo de granularidad múltiple.
- Marca temporal:
  - Reloj del sistema.
  - Contador lógico.
  - marca-temporal-E*(Q)*.
  - marca-temporal-L*(Q)*.
- Protocolo de ordenación por marcas temporales:
  - Regla de escritura de Thomas.
- Protocolos basados en validación:
  - Fase de lectura.
  - Fase de validación.
  - Fase de escritura.
  - Comprobación de validación.
- Ordenación por marcas temporales multiversión.
- Bloqueo de dos fases multiversión:
  - Transacciones de solo lectura.
  - Transacciones de actualización.
- Aislamiento de instantáneas:
  - Actualización perdida.
  - Primer compromiso gana.
  - Primera actualización gana.
  - Atasco de escritura.
  - Selección para actualización.
- Operaciones para insertar y borrar.
- Fenómeno fantasma.
- Protocolo de bloqueo de índices.
- Bloqueo de predicados.
- Niveles débiles de consistencia:
  - Consistencia de grado dos.
  - Estabilidad del cursor.
- Control de concurrencia optimista sin validación de lectura.
- Conversaciones.
- Concurrencia en índices:
  - Cangrejo.
  - Árboles B enlazados.
  - Protocolo de bloqueo con árboles B enlazados.
  - Bloqueo de la clave siguiente.

Ejercicios prácticos

15.1. Demuestre que el protocolo de bloqueo de dos fases asegura la secuencialidad en cuanto a conflictos y que se pueden secuenciar las transacciones a través de sus puntos de bloqueo.

15.2. Considere las dos transacciones siguientes:

T_{34}: leer*(A)*;
leer*(B)*;
si $A = 0$ **entonces** $B := B + 1$;
escribir*(B)*.

T_{35}: leer*(B)*;
leer*(A)*;
si $B = 0$ **entonces** $A := A + 1$;
escribir*(A)*.

Añada a las transacciones T_{34} y T_{35} las instrucciones de bloqueo y desbloqueo para que sigan el protocolo de bloqueo de dos fases. ¿Puede producir la ejecución de estas transacciones un interbloqueo?

15.3. ¿Qué ventajas proporciona el bloqueo riguroso de dos fases? Compárelo con otras formas de bloqueo de dos fases.

15.4. Considere una base de datos organizada como un árbol con raíz. Suponga que se inserta un nodo ficticio entre cada par de nodos. Demuestre que, si se sigue el protocolo de árbol con este nuevo árbol, se obtiene mayor concurrencia que con el árbol original.

15.5. Demuestre con un ejemplo que hay planificaciones que son posibles con el protocolo de árbol que no lo son con otros protocolos de bloqueo de dos fases, y viceversa.

15.6. Considere la siguiente extensión del protocolo de bloqueo de árbol que permite bloqueos compartidos y exclusivos:

- Una transacción puede ser de solo lectura, en cuyo caso solo puede solicitar bloqueos compartidos, o bien puede ser de actualización, en cuyo caso solo puede solicitar bloqueos exclusivos.
- Las transacciones deben seguir las reglas del protocolo de árbol. Las transacciones de solo lectura deben bloquear primero cualquier elemento de datos, mientras que las transacciones de actualización deben bloquear primero la raíz.

Demuestre que este protocolo asegura la secuencialidad y la ausencia de interbloqueos.

15.7. Considere el siguiente protocolo de bloqueo basado en grafos, que solo permite bloqueos exclusivos y que funciona con grafos de datos con forma de grafo dirigido acíclico con raíz.

- Una transacción puede bloquear en primer lugar cualquier nodo.
- Para bloquear cualquier otro nodo, la transacción debe poseer un bloqueo sobre la mayoría de los padres de dicho nodo.

Demuestre que este protocolo asegura la secuencialidad y la ausencia de interbloqueos.

15.8. Considere el siguiente protocolo de bloqueo basado en grafos que solo permite bloqueos exclusivos y que funciona con grafos de datos con forma de grafo dirigido acíclico con raíz.

- Una transacción puede bloquear en primer lugar cualquier nodo.
- Para bloquear cualquier otro nodo, la transacción debe haber visitado a todos los padres de dicho nodo y debe poseer un bloqueo sobre uno de los padres del vértice.

Demuestre que este protocolo asegura la secuencialidad y la ausencia de interbloqueos.

15.9. El bloqueo no se hace explícitamente en lenguajes de programación persistentes. En su lugar se deben bloquear los objetos (o sus páginas correspondientes) cuando se accede a dichos objetos. Muchos de los sistemas operativos más modernos permiten al usuario definir protecciones de acceso (sin acceso, lectura, escritura) para las páginas, y aquellos accesos a la memoria que violen las protecciones de acceso dan como resultado una violación de protección (véase la orden mprotect de Unix, por ejemplo).

Describa la forma en que se puede usar el mecanismo de protección de acceso para bloqueos a nivel de página en lenguajes de programación persistentes.

15.10. Considere una base de datos que tiene la operación atómica **incrementar,** además de las operaciones leer y escribir. Sea V el valor del elemento de datos X.

La operación:

incrementar*(X)* en C

asigna el valor $V + C$ a X en un paso atómico. El valor de X no está disponible hasta que no se ejecute posteriormente una operación leer(X). En la Figura 15.23 se muestra una matriz de compatibilidad de bloqueos para tres tipos de bloqueo: modo compartido, exclusivo y de incremento.

a. Demuestre que, si todas las transacciones bloquean el dato al que acceden en el modo correspondiente, entonces el bloqueo de dos fases asegura la secuencialidad.

b. Demuestre que la inclusión del bloqueo en modo **incrementar** permite una mayor concurrencia. *Sugerencia*: considérense las transacciones de transferencia de fondos del ejemplo bancario.

	C	X	I
C	cierto	falso	falso
X	falso	falso	falso
I	falso	falso	cierto

Figura 15.23. Matriz de compatibilidad de bloqueos.

15.11. En la ordenación por marcas temporales, **marca-temporal-E***(Q)* indica la mayor marca temporal de todas las transacciones que hayan ejecutado escribir*(Q)* con éxito. Suponga que en lugar de ello, **marca-temporal-E***(Q)* se define como la marca temporal de la transacción más reciente que haya ejecutado escribir*(Q)* con éxito.

¿Hay alguna diferencia al cambiar esta definición?

Razone la respuesta.

15.12. La utilización de un bloqueo de granularidad múltiple puede necesitar más o menos bloqueos que en un sistema equivalente con una granularidad simple de bloqueo. Indique ejemplos de ambas situaciones y compare el aumento relativo de la concurrencia que se permite.

15.13. Considere el esquema de control de concurrencia basado en la validación de la Sección 15.5. Demuestre que si se elige Validación(T_i) en lugar de Inicio(T_i) como marca temporal de la transacción T_i, se puede esperar un mejor tiempo de respuesta debido a que la tasa de conflictos entre las transacciones es realmente baja.

15.14. Para cada uno de los protocolos siguientes, describa los aspectos de aplicación práctica que sugieran utilizar el protocolo y los aspectos que sugieran no usarlo:

- Bloqueo de dos fases.
- Bloqueo de dos fases con granularidad múltiple.
- Protocolo de árbol.
- Ordenación por marcas temporales.
- Validación.
- Ordenación por marcas temporales multiversión.
- Bloqueo de dos fases multiversión.

15.15. Explique por qué la siguiente técnica de ejecución de transacciones puede proporcionar mayor rendimiento que la utilización del bloqueo estricto de dos fases: primero se ejecuta la transacción sin adquirir ningún bloqueo y sin realizar ninguna escritura en la base de datos, como en las técnicas basadas en validación, pero a diferencia de las técnicas de validación, no se realiza otra validación o escritura en la base de datos. En cambio, se vuelve a ejecutar la transacción utilizando el bloqueo estricto de dos fases. *Sugerencia*: considere esperas para la E/S de disco.

15.16. Considere el protocolo de ordenación por marcas temporales, y dos transacciones, una que escribe dos elementos de datos p y q, y otra que lee los mismos dos elementos de datos. Obtenga una planificación por medio de la cual la comprobación por marcas temporales para una operación escribir falle y provoque el reinicio de la primera transacción, causando a su vez una cancelación en cascada de la otra transacción.

Demuestre cómo podría acabar en inanición de las dos transacciones. (Tal situación, en la que dos o más procesos realizan acciones pero no se puede completar la tarea porque se interacciona con otros procesos, se denomina **interbloqueo**).

15.17. Diseñe un protocolo basado en marcas temporales que evite el fenómeno fantasma.

15.18. Suponga que se utiliza el protocolo de árbol de la Sección 15.1.5 para administrar el acceso concurrente a un árbol B^+. Puesto que puede haber una división en una inserción que afecte a la raíz, se deduce que una operación inserción no puede liberar ningún bloqueo hasta que no se complete la operación entera. ¿Bajo qué circunstancias es posible liberar antes un bloqueo?

15.19. El protocolo de aislamiento de instantánea usa un paso de validación en el que la transacción T antes de realizar una escritura de un elemento de datos comprueba si alguna transacción concurrente con T ya ha escrito ese elemento de dato.

a. Una implementación directa usa para cada transacción una marca de tiempo de inicio y una marca de tiempo de comprometida, además de un *conjunto de actualización*, que es el conjunto de elementos de datos que actualiza la transacción.

Explique cómo realizar la validación para el esquema primer-compromiso-gana utilizando las marcas de tiempo de la transacción junto con el conjunto de actualizaciones. Se puede suponer que los pasos de validación y otros procesos de comprometida se ejecutan secuencialmente, es decir, una transacción tras otra.

b. Explique cómo se puede implementar el paso de validación como parte del proceso de compromiso en el esquema primer-compromiso-gana usando una modificación del esquema anterior, en el que en lugar de utilizar conjuntos de actualizaciones, cada elemento de datos tiene asociada una marca de tiempo. Se puede suponer, de nuevo, que los pasos de validación y otros procesos de compromiso se ejecutan secuencialmente.

c. El esquema primera-actualización-gana se puede implementar utilizando marcas de tiempo, como se describe anteriormente, excepto que la validación se realiza inmediatamente tras adquirir un bloque exclusivo, en lugar de realizarse en el momento del compromiso.

i. Explique cómo asignar marcas de tiempo a los elementos de datos para implementar el esquema primera-actualización-gana.

ii. Demuestre que como resultado del bloqueo si se repite la validación en el momento de compromiso el resultado no debería cambiar.

iii. Explique por qué no se necesita realizar, en este caso, los pasos de validación y otros procesos de compromiso secuencialmente.

Ejercicios

15.20. ¿Qué ventajas proporciona el bloqueo estricto de dos fases? ¿Qué inconvenientes tiene?

15.21. Muchas implementaciones de sistemas de bases de datos utilizan el bloqueo estricto de dos fases. Indique tres razones que expliquen la popularidad de este protocolo.

15.22. Considere una variante del protocolo de árbol llamada protocolo de *bosque*. La base de datos está organizada como un bosque de árboles con raíz. Cada transacción T_i debe seguir las siguientes reglas:

- El primer bloqueo en un árbol puede hacerse sobre cualquier elemento de datos.
- Se pueden solicitar el segundo y posteriores bloqueos solo si el padre del nodo solicitado está actualmente bloqueado.
- Se pueden desbloquear los elementos de datos en cualquier momento.
- T_i no puede volver a bloquear un elemento de datos después de haberlo desbloqueado.

Demuestre que el protocolo de bosque *no* asegura la secuencialidad.

15.23. ¿Bajo qué condiciones es menos costoso evitar los interbloqueos que permitirlos y luego detectarlos?

15.24. Si se evitan los interbloqueos, ¿sigue siendo posible que haya inanición? Razone la respuesta.

15.25. En el protocolo de granularidad múltiple, ¿qué diferencia hay entre bloqueo implícito y explícito?

15.26. Aunque el modo IXC es útil para el bloqueo de granularidad múltiple, no se usa un modo exclusivo e intencional-compartido (ICX). ¿Por qué no es útil?

15.27. Las reglas del protocolo de granularidad múltiple indican que una transacción T_i puede bloquear un nodo Q en el modo C o en el modo IC solo si actualmente T_i ha bloqueado al padre de Q en los modos IX o IC. Teniendo en cuenta que los bloqueos IXC y C son más fuertes que los bloqueos IX e IC, ¿por qué no permite el protocolo bloquear un nodo en los modos C o IC si el padre tiene un bloqueo en el modo IXC o C?

15.28. Cuando una transacción retrocede en el protocolo de ordenación por marcas temporales se le asigna una nueva marca temporal. ¿Por qué no puede conservar simplemente su antigua marca temporal?

15.29. Demuestre que hay planificaciones que son posibles con el protocolo de bloqueo de dos fases que no lo son con el protocolo de marcas temporales, y viceversa.

15.30. En una versión modificada del protocolo de marcas temporales se necesita comprobar un bit de compromiso para saber si una petición de lectura debe esperar o no. Explique cómo se puede evitar con el bit de compromiso que aborten en cascada. ¿Por qué no se necesita hacer esta comprobación con las peticiones de escritura?

15.31. Como se ha tratado en el Ejercicio práctico 15.19, el aislamiento de instantáneas se puede implementar usando una forma de validación de marcas de tiempo. Sin embargo, al contrario que con el esquema de ordenación de marcas temporales, que garantiza la secuencialidad, el aislamiento de instantáneas no lo garantiza. Explique cuál es la diferencia clave entre estos protocolos que genera esta diferencia.

15.32. Describa las similitudes y diferencias clave entre la implementación basada en marcas de tiempo de la versión primer-compromiso-gana del aislamiento de instantáneas, descrito en el Ejercicio práctico 15.19, y el control de concurrencia optimista sin esquema de validación de lectura, que se describe en la Sección 15.9.3.

15.33. Explique el fenómeno fantasma. ¿Por qué produce este fenómeno una ejecución concurrente incorrecta a pesar de utilizar el protocolo de bloqueo de dos fases?

15.34. Explique la razón por la cual se utiliza la consistencia de grado dos. ¿Qué inconvenientes tiene esta técnica?

15.35. Indique ejemplos de planificaciones para demostrar que con el bloqueo de valores clave si alguna búsqueda, inserción o borrado no bloquea el siguiente valor clave, el fenómeno fantasma podría ser indetectable.

15.36. Suponga que muchas transacciones modifican un elemento común (por ejemplo, el saldo de una sucursal) y elementos privados (por ejemplo, saldos de cuentas en concreto). Explique cómo se puede aumentar el grado de concurrencia (y por tanto el rendimiento) ordenando las operaciones de la transacción.

15.37. Considere el siguiente protocolo de bloqueo: todos los elementos se numeran y, cada vez que un elemento se desbloquea, solo se pueden bloquear elementos con un número mayor. Los bloqueos se pueden liberar en cualquier momento. Solo se usan bloqueos X.

Demuestre mediante un ejemplo que este bloqueo no garantiza la secuencialidad.

Notas bibliográficas

Gray y Reuter [1993] proporcionan un libro de texto detallado que cubre conceptos de procesamiento de transacciones, incluyendo conceptos de control de concurrencia y detalles de implementación. Bernstein y Newcomer [1997] proporcionan un libro de texto que trata varios aspectos del procesamiento de transacciones, incluyendo el control de concurrencia.

Eswaran et ál. [1976] introdujeron el protocolo de bloqueo de dos fases. El protocolo de bloqueo de árbol es de Silberschatz y Kedem [1980]. Yannakakis et ál. [1979], Kedem y Silberschatz [1983], y Buckley y Silberschatz [1985] desarrollaron otros protocolos de bloqueo que no son de dos fases y que operan con grafos más generales. Korth [1983] explora varios modos de bloqueo que se pueden obtener a partir de los modos básico, compartido y exclusivo.

El Ejercicio práctico 15.4 es de Buckley y Silberschatz [1984]. El Ejercicio práctico 15.6 es de Kedem y Silberschatz [1983]. El Ejercicio práctico 15.7 es de Kedem y Silberschatz [1979]. El Ejercicio práctico 15.8 es de Yannakakis et ál. [1979]. El Ejercicio práctico 15.10 es de Korth [1983].

El protocolo de bloqueo para elementos de datos de granularidad múltiple es de Gray et ál. [1975]. Gray et ál. [1976] presentan una descripción detallada. Kedem y Silberschatz [1983] formalizan el bloqueo de granularidad múltiple para una colección arbitraria de modos de bloqueo (que permite más semántica que simplemente leer y escribir). Este enfoque incluye una clase de modos de bloqueo llamados modos de *actualización* para permitir la conversión de bloqueos. Carey [1983] extiende la idea de granularidad múltiple a la de control de concurrencia basado en marcas temporales. Korth [1982] presenta una extensión del protocolo que asegura la ausencia de interbloqueos.

El esquema de control de concurrencia basado en marcas temporales es de Reed [1983]. Buckley y Silberschatz [1983] presentan un algoritmo de marcas temporales que no necesita retroceso para asegurar la secuencialidad. El esquema de control de concurrencia basado en validación es de Kung y Robinson [1981].

Reed [1983] introdujo la ordenación por marcas temporales multiversión. En Silberschatz [1982] aparece un algoritmo de bloqueo de árbol multiversión.

En Gray et ál. [1975] se introduce la consistencia de grado dos. Los niveles de consistencia —o aislamiento— que ofrece SQL se explican y comentan en Berenson et ál. [1995]. Muchos sistemas comerciales de bases de datos usan enfoques basados en versiones en combinación con el bloqueo. Oracle y SQL Server usa una forma de aislamiento basada en instantáneas según el protocolo descrito en la Sección 15.6.2. Los detalles se encuentran en los Capítulos 27, 28 y 30, respectivamente.

Hay que tener en cuenta que en PostgreSQL (en la versión 8.1.4) y en Oracle (en la versión 10g), al establecer el nivel de aislamiento en secuenciable se usa el aislamiento de instantánea, lo que no garantiza la secuencialidad. Fekete et ál. [2005] describen cómo asegurar las ejecuciones secuenciales bajo aislamiento de instantáneas reescribiendo ciertas transacciones para introducir conflictos; estos conflictos aseguran que las transacciones no se pueden ejecutar concurrentemente en aislamiento de instantáneas; Jorwekar et ál. [2007] describen un enfoque en el que dado un conjunto de transacciones (parametrizadas) que ejecutan bajo aislamiento de instantáneas, se puede comprobar si las transacciones son vulnerables a no ser secuenciables.

Bayer y Schkolnick [1977], y Johnson y Shasha [1993] estudiaron la concurrencia en árboles B^+. La técnica que se ha presentado en la Sección 15.10 está basada en Kung y Lehman [1980], y en Lehman y Yao [1981]. En Mohan [1990a], y Mohan y Levine [1992] se describe la técnica del bloqueo del valor clave utilizada en ARIES, que proporciona una gran concurrencia en el acceso a los árboles B^+. Ellis [1987] presenta una técnica de control de concurrencia para asociación lineal.

Sistema de recuperación 16

Las computadoras, al igual que cualquier otro dispositivo, están sujetas a fallos debidos a diferentes motivos, como: fallos de disco, cortes de corriente, errores en el software, un incendio en la habitación donde se encuentra la computadora o incluso un sabotaje. En cada uno de estos casos puede perderse información. Por tanto, el sistema de bases de datos debe realizar con anticipación acciones que garanticen que las propiedades de atomicidad y durabilidad de las transacciones, presentadas en el Capítulo 14, se mantienen a pesar de tales fallos. Una parte integral de un sistema de bases de datos es el **esquema de recuperación,** responsable de restaurar la base de datos al estado consistente previo al fallo. El esquema de recuperación también debe proporcionar **alta disponibilidad;** es decir, debe minimizar el tiempo durante el cual la base de datos no se puede usar después de un fallo.

16.1. Clasificación de los fallos

En un sistema pueden producirse varios tipos de fallos, cada uno de los cuales requiere un tratamiento diferente. En este capítulo solo se consideran los siguientes tipos de fallos:

- **Fallo en la transacción.** Hay dos tipos de errores que pueden hacer que una transacción falle:
  - **Error lógico.** La transacción no puede continuar con su ejecución normal a causa de alguna condición interna, como una entrada incorrecta, datos no encontrados, desbordamiento o exceso del límite de recursos.
  - **Error del sistema.** El sistema se encuentra en un estado no deseado (por ejemplo, de interbloqueo) como consecuencia del cual una transacción no puede continuar con su ejecución normal. La transacción, sin embargo, se puede volver a ejecutar más tarde.
- **Caída del sistema.** Un mal funcionamiento del hardware o un error en el software de la base de datos o del sistema operativo causan la pérdida del contenido de la memoria volátil y abortan el procesamiento de una transacción. El contenido de la memoria no volátil permanece intacto y no se corrompe.

  La suposición de que los errores de hardware o software fuercen una parada del sistema, pero no corrompan el contenido de la memoria no volátil, se conoce como **supuesto de fallo-parada.** Los sistemas bien diseñados tienen numerosas comprobaciones internas, tanto en el hardware como en el software, que abortan el sistema cuando existe un error. De ahí que el supuesto de fallo-parada sea razonable.
- **Fallo de disco.** Un bloque del disco puede perder su contenido como resultado de una colisión de la cabeza lectora, o de un fallo durante una operación de transferencia de datos. Las copias de los datos que se encuentran en otros discos o en archivos de seguridad en medios de almacenamiento secundarios, como cintas, se utilizan para recuperarse del fallo.

Para determinar el medio por el que el sistema debe recuperarse de los fallos es necesario identificar los modos de fallo de los dispositivos de almacenamiento. A continuación se verá cómo afectan estos modos de fallo al contenido de la base de datos. Por tanto, se pueden proponer algoritmos para garantizar la consistencia de la base de datos y la atomicidad de las transacciones a pesar de los fallos. Estos algoritmos, conocidos como algoritmos de recuperación, constan de dos partes:

1. Acciones llevadas a cabo durante el procesamiento normal de transacciones para asegurar que existe información suficiente para permitir la recuperación frente a los fallos.
2. Acciones llevadas a cabo después de ocurrir un fallo para restablecer el contenido de la base de datos a un estado que asegure la consistencia de la base de datos, la atomicidad de la transacción y la durabilidad.

16.2. Almacenamiento

Como se trató en el Capítulo 10, los diferentes elementos que componen una base de datos se pueden almacenar y posteriormente acceder a ellos en diferentes medios de almacenamiento. En la Sección 14.3 se trataron los medios de almacenamiento, que se pueden distinguir por su velocidad, capacidad y resistencia relativa a los fallos. Se identificaron tres categorías de almacenamiento:

- **Almacenamiento volátil.**
- **Almacenamiento no volátil.**
- **Almacenamiento estable.**

El almacenamiento estable, o de forma más precisa, una aproximación al mismo, juega un papel esencial en los algoritmos de recuperación.

16.2.1. Implementación de almacenamiento estable

Para implementar el almacenamiento estable se debe replicar la información necesaria en varios medios de almacenamiento no volátil (normalmente discos) con modos de fallo independientes, y actualizar esa información de manera controlada para asegurar que si se produce un fallo durante una transferencia de datos este no dañará la información necesaria.

Recuerde (del Capítulo 10) que los sistemas RAID garantizan que el fallo de un solo disco (incluso durante una transferencia de datos) no conduce a la pérdida de los datos. La variante más sencilla y rápida de RAID es el disco con imagen, que guarda dos copias de cada bloque en distintos discos. Otras formas de RAID ofrecen menores costes a expensas de un rendimiento inferior.

Los sistemas RAID, sin embargo, no pueden proteger contra las pérdidas de datos debidas a desastres tales como un incendio o una inundación. Muchos sistemas de almacenamiento guardan copias de seguridad de las cintas en otro lugar como protección frente a tales desastres. No obstante, como las cintas no se pueden trasladar con-

tinuamente a otro lugar, los cambios que se hayan realizado desde el último traslado de las cintas se perderán en caso de que se produzca un desastre. Los sistemas más seguros guardan una copia de cada bloque de almacenamiento estable en un lugar remoto, escribiéndola por la red de datos, además de almacenar el bloque en un sistema de discos locales. Como los bloques se envían al sistema remoto al mismo tiempo y de la misma forma que se guardan en almacenamiento local, una vez que una operación de este tipo se completa los bloques copiados no pueden perderse, incluso en caso de que ocurriese un desastre como un incendio o una inundación. En la Sección 16.9 se estudian estos sistemas de *copia de seguridad remota*.

En el resto de esta sección se estudia la manera de proteger los medios de almacenamiento de los errores durante una transferencia de datos. Las transferencias de bloques entre la memoria y el disco pueden acabar de diferentes formas:

- **Éxito.** La información transferida llega a su destino con seguridad.
- **Fallo parcial.** Se produce un fallo en medio de la transferencia, y al bloque de destino llega información incorrecta.
- **Fallo total.** El fallo se produce lo bastante pronto en la transferencia como para que el bloque de destino permanezca intacto.

Es necesario que, si se produce un **fallo de una transferencia de datos,** el sistema lo detecte e invoque a un procedimiento de recuperación para restaurar el bloque a un estado estable. Para ello, el sistema debe mantener dos bloques físicos por cada bloque lógico de la base de datos; en el caso de los discos con imagen, ambos bloques están en el mismo lugar; en el caso de la copia de seguridad remota, uno de los bloques es local mientras que el otro está en un lugar remoto. La operación de salida se ejecuta de la siguiente manera:

1. Se escribe la información en el primer bloque físico.
2. Cuando la primera escritura se completa correctamente, se escribe la misma información en el segundo bloque físico.
3. La salida solo está completa después de que la segunda escritura finalice correctamente.

Si el sistema falla mientras se escriben los bloques, es posible que las dos copias sean inconsistentes una respecto a la otra. Durante la recuperación se examina cada par de bloques físicos. Si ambos coinciden y no existe ningún error detectable, entonces no son necesarias más acciones (recuerde que los errores de los bloques del disco, como puede ser una escritura parcial del bloque, se detectan mediante el almacenamiento de una suma de control en cada bloque). Si un bloque contiene un error detectable, se reemplaza su contenido por el del segundo bloque. Si ninguno de los dos bloques contiene errores detectables, pero su contenido es diferente, el sistema sustituye el contenido del primer bloque por el valor del segundo. Este procedimiento de recuperación garantiza que la escritura en almacenamiento estable o bien se completa correctamente (es decir, se actualizan todas las copias) o bien no produce ningún cambio.

El requisito de comparar cada par correspondiente de bloques durante la recuperación es bastante costoso. Puede reducirse considerablemente ese coste si se registran las escrituras de bloques que están en progreso utilizando una pequeña cantidad de RAM no volátil. Durante la recuperación solamente es necesario comparar aquellos bloques en los que se estuviera escribiendo.

Los protocolos para escribir un bloque en un lugar remoto son similares a los utilizados para escribir bloques en un sistema de disco con imagen, que se trataron en el Capítulo 10 y, en particular, en el Ejercicio práctico 10.3.

Este procedimiento puede extenderse fácilmente para permitir el uso de un número arbitrariamente alto de copias de cada bloque de almacenamiento estable. Aunque un número elevado de copias reduce la probabilidad de fallo incluso por debajo de la conseguida con dos copias, habitualmente es razonable la simulación de almacenamiento estable con solo dos copias.

16.2.2. Acceso a los datos

Como se vio en el Capítulo 10, el sistema de bases de datos reside permanentemente en almacenamiento no volátil (normalmente discos) y solo partes de la base de datos se encuentran en la memoria en un momento dado.[1] La base de datos se divide en unidades de almacenamiento de longitud fija denominadas **bloques.** Los bloques son las unidades de datos que se transfieren desde y hacia el disco y pueden contener varios elementos de datos. Se supone que ningún elemento de datos ocupa dos o más bloques. Esta suposición es realista para la mayoría de las aplicaciones de procesamiento de datos tales como los ejemplos de un banco o una universidad.

Las transacciones realizan la entrada de información desde el disco hacia la memoria principal y luego llevan a cabo la salida devolviendo la información al disco. Las operaciones de entrada y salida se realizan en unidades de bloque. Nos referiremos a los bloques que residen en el disco como **bloques físicos,** y a los que residen temporalmente en la memoria principal como **bloques de memoria intermedia.** El área de memoria en la que los bloques residen temporalmente se denomina **memoria intermedia de disco.**

Las transferencias de un bloque entre disco y memoria principal comienzan mediante las dos operaciones siguientes:

1. entrada(B) transfiere el bloque físico B a la memoria principal.
2. salida(B) transfiere el bloque de memoria intermedia B al disco y reemplaza allí al correspondiente bloque físico.

Este esquema se muestra en la Figura 16.1.

Conceptualmente, cada transacción T_i posee un área de trabajo privada en la cual se guardan copias de todos los elementos de datos a los que ha accedido y actualizado. Este área de trabajo se crea cuando se comienza una transacción y se elimina cuando la transacción o bien se compromete o bien aborta. El elemento de datos X almacenado en el área de trabajo de la transacción T_i se denotará como x_i. La transacción T_i interactúa con el sistema de bases de datos por medio de transferencias de datos desde su área de trabajo hacia la memoria intermedia del sistema y viceversa. Las transferencias de datos se realizarán utilizando las dos operaciones siguientes:

1. leer(X) asigna el valor del elemento de datos X a la variable local x_i. Esta operación se ejecuta de la siguiente forma:
   a. Si el bloque B_X en el que reside X no está en la memoria principal, entonces se ejecuta entrada(B_X).
   b. Asigna a x_i el valor de X del bloque de memoria intermedia.
2. escribir(X) asigna el valor de la variable local x_i al elemento de datos X en el bloque de memoria intermedia. Esta operación se ejecuta de la siguiente forma:
   a. Si el bloque B_X en el que reside X no está en la memoria principal, entonces se ejecuta entrada(B_X).
   b. Asigna el valor de x_i a X en la memoria intermedia B_X.

Observe que ambas operaciones pueden requerir la transferencia de un bloque desde el disco a la memoria principal. En cambio, ninguna de ellas requiere específicamente la transferencia de un bloque desde la memoria principal al disco.

El bloque de memoria intermedia se escribe en el disco bien porque el gestor de la memoria intermedia necesita espacio en la memoria para otros propósitos, o bien porque el sistema de base de datos desea reflejar en el disco los cambios en B. Se dice que el sistema de bases de datos **fuerza la salida** de la memoria intermedia B si ejecuta el comando salida(B).

1 Existe una categoría especial de sistemas de bases de datos, denominada *sistemas de bases de datos en memoria principal*, en la que toda la base de datos puede cargarse en memoria de una sola vez. Estos sistemas se verán en la Sección 26.4.

Cuando una transacción necesita acceder a un elemento de datos X por primera vez, debe ejecutar leer(X). Posteriormente, todas las actualizaciones de X se llevan a cabo en x_i. En cualquier momento durante su ejecución una transacción puede ejecutar escribir(X) para reflejar los cambios de X en la base de datos; escribir(X) debe realizarse siempre después de la última escritura en X.

La operación salida(B_X) sobre el bloque de memoria intermedia B_X en el que reside X no tiene por qué tener efecto inmediatamente después de ejecutar escribir(X), ya que el bloque B_X puede contener otros elementos de datos a los que aún se esté accediendo. Por tanto, la salida real puede realizarse más tarde. Observe que, si el sistema falla después de ejecutar la operación escribir(X), pero antes de ejecutar salida(B_X), no se escribe nunca en el disco el nuevo valor de X y, por tanto, se pierde. Como se verá en breve, el sistema de base de datos ejecuta acciones adicionales para asegurar que las actualizaciones realizadas por transacciones comprometidas no se pierden incluso aunque se produzca un fallo del sistema.

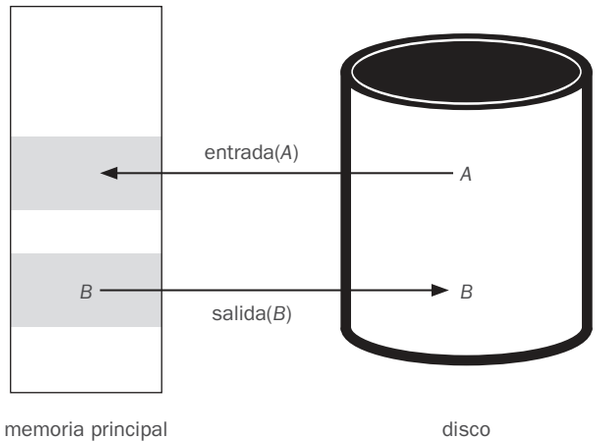


Figura 16.1. Operaciones de almacenamiento de bloques.

16.3. Recuperación y atomicidad

Considere de nuevo el sistema bancario simplificado y una transacción T_i que transfiere 50 € desde la cuenta A a la cuenta B, siendo los saldos iniciales de A y de B de 1.000 € y 2.000 €, respectivamente. Suponga que el sistema cae durante la ejecución de T_i después de haberse ejecutado salida(B_A), pero antes de la ejecución de salida(B_B), donde B_A y B_B denotan los bloques de memoria intermedia en los que residen A y B. Al perderse el contenido de la memoria, no se sabe el resultado de la transacción.

Cuando el sistema se reinicia, el valor de A podría ser de 950 €, mientras que el de B podría ser de 2.000 €, lo que es claramente inconsistente con los requisitos de atomicidad de la transacción T_i. Desafortunadamente no existe forma de descubrir, examinando el estado de la base de datos, qué bloques han conseguido salir y cuáles no antes del fallo del sistema. Es posible que la transacción se haya completado, actualizando la transacción en almacenamiento estable desde el estado inicial en que los valores de A y B eran 1.000 € y 1.950 €; también es posible que la transacción no cambiase el almacenamiento estable y los valores de A y B fuesen inicialmente de 950 € y 2.000 €; o que la actualización de B se completase pero no la de A; o que la actualización de A se completase pero no la de B.

El objetivo es realizar todos los cambios inducidos por T_i o no llevar a cabo ninguno. Sin embargo, si T_i realiza varias modificaciones en la base de datos, pueden necesitarse varias operaciones de salida y puede producirse un fallo después de haber concluido alguna de estas modificaciones, pero antes de haber terminado todas.

Para conseguir el objetivo de la atomicidad primero se debe efectuar la operación de salida de la información, que describe las modificaciones en el almacenamiento estable sin modificar todavía la base de datos. Como se verá, este procedimiento permitirá asegurar que todas las modificaciones realizadas por transacciones comprometidas quedan reflejadas en la base de datos (quizás durante las acciones de recuperación después de un fallo). Esta información también ayuda a asegurar que no se realizan modificaciones que se mantengan en la base de datos de ninguna transacción abortada.

16.3.1. Registro histórico

La estructura más ampliamente utilizada para guardar las modificaciones de una base de datos es el **registro histórico.** El registro histórico es una secuencia de **registros** que almacena todas las actividades de actualización de la base de datos.

Existen varios tipos de registros del registro histórico. Un **registro de actualización del registro histórico** describe una única escritura en la base de datos y tiene los siguientes campos:

- El **identificador de la transacción** es un identificador único de la transacción que realiza la operación escribir.
- El **identificador del elemento de datos** es un identificador único del elemento de datos que se escribe. Normalmente suele coincidir con la ubicación del elemento de datos en el disco, que consta del identificador de bloque en que reside en el disco, y el desplazamiento dentro del bloque.
- El **valor anterior** es el valor que tenía el elemento de datos antes de la escritura.
- El **valor nuevo** es el valor que tendrá el elemento de datos después de la escritura.

Se representa una actualización del registro histórico como $<T_i, X_j, V_1, V_2>$, indicando que la transacción T_i ha realizado una escritura sobre el elemento de datos X_j. X_j tenía el valor V_1 antes de la escritura y tendrá el valor V_2 después de la escritura. Existen otros registros especiales para registrar sucesos significativos durante el procesamiento de una transacción, tales como el comienzo de una transacción y el compromiso o aborto de la misma. Entre los tipos de registros del registro histórico están:

- $<T_i$ iniciada$>$. La transacción T_i ha comenzado.
- $<T_i$ comprometida$>$. La transacción T_i se ha comprometido.
- $<T_i$ abortada$>$. La transacción T_i ha sido abortada.

Más adelante se incluirán otros tipos de registros.

Cuando una transacción realiza una escritura es fundamental que se cree el registro del registro histórico correspondiente a esa escritura antes de modificar la base de datos. Una vez que el registro del registro histórico existe, se puede realizar la salida de la modificación a la base de datos si se desea. Además, es posible *deshacer* una modificación que ya haya salido a la base de datos. Se deshará utilizando el campo valor-anterior de los registros del registro histórico.

Para que los registros del registro histórico sean útiles para recuperarse frente a errores del disco o del sistema, el registro histórico debe residir en almacenamiento estable. Por ahora se supondrá que cada registro del registro histórico se escribe, tan pronto como se crea, al final del registro histórico en almacenamiento estable. En la Sección 16.5 se verán las condiciones necesarias para poder relajar este requisito de forma segura de modo que se reduzca la sobrecarga impuesta por el registro histórico. Observe que en el registro histórico se tiene constancia de todas las actividades de la base de datos. Como consecuencia, el tamaño de los datos almacenados en el registro histórico puede llegar a ser extremadamente grande. En la Sección 16.3.6 se mostrará bajo qué condiciones se puede borrar información del registro histórico de manera segura.

COPIAS Y PÁGINAS EN LA SOMBRA

En el esquema de **copia en la sombra,** una transacción que quiere actualizar la base de datos primero crea una copia completa de la misma. Todas las actualizaciones se realizan en la nueva copia de la base de datos, dejando la copia original, la **copia en la sombra,** sin tocar. Si en cualquier momento la transacción aborta, el sistema simplemente elimina la nueva copia. La antigua copia de la base de datos no se ve afectada. La copia actual de la base de datos queda identificada por un apuntador denominado apuntador-db, que se almacena en el disco.

Si la transacción resulta parcialmente comprometida (es decir, ejecuta su sentencia final) se compromete de la siguiente forma: en primer lugar, se pregunta al sistema operativo para asegurarse de que se han escrito en disco todas las páginas de la nueva copia de la base de datos. (En los sistemas Unix se utiliza el comando fsync). Una vez que el sistema operativo ha escrito todas las páginas en el disco, el sistema de bases de datos actualiza el puntero-db para que apunte a la nueva copia de la base de datos; la nueva copia se convierte en la copia actual. La copia antigua se elimina. Se dice que la transacción está *comprometida* en el momento en que el puntero-db actualizado se escribe en el disco.

La implementación depende, realmente, de que la escritura del puntero-db sea atómica; es decir, o se escriben todos sus bytes o no se escribe ninguno. Los sistemas de disco proporcionan actualizaciones atómicas de bloques completos, o al menos de un sector del disco. En otras palabras, el sistema de disco garantiza que el puntero-db se actualizará atómicamente, siempre que se asegure que el mismo se encuentra en un único sector, lo que se puede garantizar almacenando el puntero-db al inicio de un bloque.

Los esquemas de copia en la sombra suelen utilizarlos los editores de texto (guardar el archivo es equivalente a comprometer la transacción, mientras que abandonar sin guardar el archivo es equivalente a abortar la transacción). La copia en la sombra se puede utilizar en bases de datos pequeñas, pero la copia de grandes bases de datos puede ser extremadamente costosa. Una variante es la denominada **paginación en la sombra,** que reduce las copias de la siguiente forma: el esquema usa una tabla de páginas que contiene punteros a todas las páginas; la propia tabla de páginas y todas las páginas actualizadas se copian a una nueva ubicación. Las páginas que no se actualizan en ninguna transacción no se copian, pero en lugar de una nueva tabla de página solo guarda un puntero a la página original. Cuando una transacción se compromete, se actualiza atómicamente el puntero a la tabla de página, que actúa como puntero-db, para apuntar a la nueva copia.

Por desgracia, la paginación en la sombra no funciona bien con las transacciones concurrentes y no se suele usar en las bases de datos.

16.3.2. Modificación de la base de datos

Como se ha indicado anteriormente, una transacción crea un registro del registro histórico antes de modificar la base de datos. Este registro permite al sistema deshacer los cambios realizados por la transacción si se tiene que abortar esta; también permite que el sistema rehaga los cambios de una transacción si se ha comprometido pero el sistema falló antes de que dichos cambios se pudieran guardar en la base de datos en disco. Para entender el papel que cumplen estos registros del registro histórico se necesita tener en cuenta los pasos que realiza una transacción cuando modifica un elemento de datos:

1. La transacción realiza algunos cálculos en su parte privada de la memoria principal.
2. La transacción modifica el bloque de datos en la memoria intermedia del disco en la memoria principal que contiene al elemento de datos.
3. El sistema de bases de datos ejecuta la operación de salida que escribe el bloque de datos en el disco.

Se dice que una transacción *modifica la base de datos* si realiza una actualización en una memoria intermedia de disco o en el propio disco; las actualizaciones en la parte privada de la memoria principal no cuentan como modificaciones de la base de datos. Si una transacción no modifica la base de datos hasta que esta comprometida, se dice que utiliza la técnica de **modificación diferida.** Si la modificación de la base de datos se produce mientras la transacción está aún activa, se dice que la transacción usa la técnica de **modificación inmediata.** La modificación diferida tiene la sobrecarga de que la transacción necesita hacer una copia local de todos los elementos de datos actualizados; más aún, si una transacción lee un elemento de datos que ha actualizado, debe leer el valor de su copia local.

Los algoritmos de recuperación que se describen en este capítulo admiten la modificación inmediata. Tal como se describen, funcionan correctamente incluso con modificación diferida, pero se pueden optimizar para reducir la sobrecarga cuando se usan con modificación diferida; se deja al lector como ejercicio.

Un algoritmo de recuperación debe tener en cuenta algunos factores, entre ellos:

- La posibilidad de que una transacción haya podido comprometerse aunque algunas modificaciones de la base de datos solo existan en la memoria intermedia del disco en memoria principal y no en la base de datos en disco.
- La posibilidad de que una transacción haya podido modificar la base de datos mientras esté en estado activo y, como consecuencia de un fallo, necesite abortar.

Como todas las modificaciones de la base de datos deben ir precedidas de la creación de un registro de registro histórico, el sistema tiene disponibles tanto el valor anterior a la modificación como el nuevo que se va a escribir en el elemento de datos. De esta forma el sistema puede realizar las operaciones de **deshacer** y **rehacer** de forma apropiada.

- **Deshacer:** utilizando el registro histórico, se ajusta el elemento de datos especificado en el registro al valor anterior.
- **Rehacer:** utilizando el registro histórico, se ajusta el elemento de datos especificado en el registro al valor nuevo.

16.3.3. Control de concurrencia y recuperación

Si el esquema de control de concurrencia permite que un elemento de datos X que ha modificado una transacción T_1 sea modificado después por otra transacción T_2 antes de que T_1 pase a comprometida, entonces deshacer los efectos de T_1 mediante la restauración del valor anterior de X (antes de que T_1 actualizase X) también desharía los efectos de T_2. Para evitar esta situación, los algoritmos de recuperación requieren que si una transacción modifica un elemento de datos, ninguna otra transacción modifique ese elemento de datos hasta que la primera pase a comprometida o abortada.

Este requisito se puede asegurar adquiriendo un bloqueo exclusivo sobre cualquier dato actualizado y manteniendo el bloqueo hasta que la transacción sea comprometida; en otras palabras, usando un bloqueo de dos fases estricto. Las técnicas de aislamiento de instantáneas y control de concurrencia basada en validación también adquieren bloqueos exclusivos sobre elementos de datos en el momento de la validación, antes de modificar los elementos de datos, y mantienen el bloqueo hasta que la transacción está comprometida; como resultado se satisface el requisito anterior incluso con esos protocolos de control de concurrencia.

Más adelante, en la Sección 16.7, se trata cómo se puede relajar el requisito anterior en algunas circunstancias.

Cuando se usa aislamiento de instantáneas o validación para el control de concurrencia, las actualizaciones de la base de datos por una transacción son (conceptualmente) diferidas hasta que la transacción está parcialmente comprometida; la técnica de modificación diferida casa de forma natural con estos esquemas de control de concurrencia. Sin embargo, hay que darse cuenta que algunas implementaciones del aislamiento de instantáneas utilizan la modificación inmediata, pero proporcionan una instantánea lógica bajo demanda: cuando una transacción necesita leer un elemento que una transacción concurrente ha actualizado, se realiza una copia del elemento (ya actualizado), y las actualizaciones hechas por transacciones concurrentes se vuelven atrás en la copia del elemento. De forma similar, la modificación inmediata de la base de datos casa de forma natural con el bloqueo de dos fases, pero la modificación diferida también se puede usar con el bloqueo de dos fases.

16.3.4. Transacción comprometida

Se dice que una transacción está **comprometida** cuando su registro de registro histórico comprometido, que es el último registro de la transacción, se ha guardado en almacenamiento estable; en este momento todos los registros de registro histórico ya se han guardado en almacenamiento estable. Por tanto, existe suficiente información como para asegurar que incluso si existe un fallo del sistema, se pueden rehacer las actualizaciones de la transacción. Si ocurre un fallo del sistema antes de que se guarde en almacenamiento estable el registro <T_i comprometida>, la transacción T_i vuelve atrás. Por tanto, la salida del bloque que contiene el registro comprometida en el registro histórico es la única acción atómica que produce que una transacción esté comprometida.[2]

Con la mayoría de las técnicas de recuperación basadas en registros históricos, incluyendo las que se describen en este capítulo, los bloques que contienen los elementos de datos modificados por una transacción no tienen por qué guardarse en almacenamiento estable cuando la transacción pasa a comprometida, pero deben guardarse un tiempo después. Se tratará este tema más adelante en la Sección 16.5.2.

16.3.5. Uso del registro histórico para rehacer y deshacer transacciones

A continuación se proporciona una descripción general de cómo se puede utilizar el registro histórico para recuperarse de un fallo del sistema y volver atrás las transacciones durante su funcionamiento normal. Sin embargo, se posponen los detalles sobre los procedimientos por recuperación ante fallos y vuelta atrás hasta la Sección 16.4.

Considere el sistema simplificado del banco. Sea T_0 una transacción que transfiere 50 € de una cuenta A a una cuenta B:

T_0: leer(A);
$A := A - 50$;
escribir(A); leer(B);
$B := B + 50$;
escribir(B).

Sea T_1 una transacción que retira 100 € de la cuenta C:

T_1: leer(C);
$C := C - 100$;
escribir(C).

El fragmento del registro histórico que contiene la información relevante de estas dos transacciones se muestra en la Figura 16.2.

En la Figura 16.3 se muestra un posible orden en el que tiene lugar la salida real tanto del sistema de bases de datos como del registro de la ejecución de T_0 y T_1.[3]

Utilizando el registro histórico, el sistema pude manejar cualquier fallo que no tenga como resultado la pérdida de la información en almacenamiento no volátil. El esquema de recuperación utiliza dos procedimientos de recuperación. Ambos utilizan el registro histórico para encontrar los datos actualizados por cada transacción T_i y los respectivos valores anterior y nuevo.

- rehacer(T_i) establece el valor de todos los elementos de datos actualizados por la transacción T_i a los valores nuevos.

  Es importante el orden en que la operación rehacer realiza las actualizaciones cuando se recupera un fallo del sistema. Si las actualizaciones se realizan en un orden diferente del que se aplicaron originalmente el estado final del elemento de datos tendrá un valor erróneo. La mayoría de los algoritmos, incluido el que se describe en la Sección 16.4, no realizan la operación rehacer independientemente, sino que realizan un recorrido por el registro histórico aplicando la operación rehacer en cada registro que encuentre. Este enfoque asegura que se mantiene el orden de las actualizaciones, y es más eficiente ya que solamente se necesita leer el registro una sola vez, en lugar de una vez por transacción.
- deshacer(T_i) restaura el valor de todos los elementos de datos actualizados por la transacción (T_i) a sus valores anteriores.

  El esquema de recuperación se describe en la Sección 16.4:
  - La operación deshacer no solo restaura los elementos de datos a su valor anterior sino que también escribe registros del registro histórico para guardar las actualizaciones realizadas como parte del proceso de deshacer. Estos registros son registros **solo-deshacer** especiales, ya que no contienen el valor anterior de los elementos de datos actualizados.

    Como con el procedimiento rehacer, el orden en que se realizan las operaciones de deshacer es importante; se verán los detalles en la Sección 16.4.
  - Cuando se completa la operación deshacer para la transacción T_i, escribe un registro <T_i aborta> indicando que se ha terminado de deshacer.

    Como se verá en la Sección 16.4, el procedimiento deshacer(T_i) solo se ejecuta una vez para cada transacción; si la transacción se vuele atrás durante su procesamiento normal o durante una recuperación de un fallo del sistema, no se encuentra ningún registro comprometida ni abortada para dicha transacción. Toda transacción, eventualmente, tendrá un registro comprometida o abortada en el registro histórico.

2 La salida de un bloque se puede convertir en atómica mediante técnicas para tratar con fallos de transferencia de datos, que se han descrito en la Sección 16.2.1.

3 Observe que este orden no se puede obtener usando la técnica de modificación diferida, porque la base de datos la modifica T_0 antes de que la transacción pase a comprometida, y de la misma forma para T_1.

Después de haberse producido un fallo del sistema, el esquema de recuperación consulta el registro histórico para determinar las transacciones que deben rehacerse y las que deben deshacerse para asegurar la atomicidad.

- Una transacción T_i debe deshacerse si el registro histórico contiene el registro <T_i iniciada>, pero no contiene el registro <T_i comprometida> ni el registro <T_i abortada>.
- Una transacción T_i debe rehacerse si el registro histórico contiene los registros <T_i iniciada> y el registro <T_i comprometida> o <T_i abortada>. Puede resultar extraño rehacer el registro T_i si aparece el registro <T_i abortada> en el registro histórico, por eso se escriben los registros solo-deshacer de las operaciones de deshacer. Por tanto, el resultado final será que en este caso se deshacen todas las modificaciones de T_i. Esta pequeña redundancia simplifica el algoritmo de recuperación y permite que el tiempo de recuperación sea más rápido.

Para ilustrarlo, considere de nuevo el ejemplo del banco con la ejecución ordenada de las transacciones T_0 y T_1, primero T_0 y después T_1. Suponga que el sistema falla antes de que se completen las transacciones. Se considerarán tres casos. El estado del registro histórico para cada uno de ellos se muestra en la Figura 16.4.

<T_0 iniciada>
<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>
<T_0 comprometida>
<T_1 iniciada>
<T_1, C, 700, 600>
<T_1 comprometida>

Figura 16.2. Fragmento del registro histórico del sistema correspondiente a T_0 y T_1.

Registro histórico	Base de datos
<T_0 iniciada>	
<T_0, A, 1000, 950>	
<T_0, B, 2000, 2050>	
	$A = 950$
	$B = 2050$
<T_0 comprometida>	
<T_1 iniciada>	
<T_1, C, 700, 600>	
	$C = 600$
<T_1 comprometida>	

Figura 16.3. Estado del registro histórico y de la base de datos correspondiente a T_0 y T_1.

(a)	(b)	(c)
<T_0 iniciada>	<T_0 iniciada>	<T_0 iniciada>
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 comprometida>	<T_0 comprometida>
	<T_1 iniciada>	<T_1 iniciada>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1 comprometida>

Figura 16.4. El mismo registro histórico mostrado en tres momentos distintos.

En primer lugar suponga que el fallo del sistema ocurre justo después de haber escrito en almacenamiento estable el registro del registro histórico para el paso:

escribir(B)

de la transacción T_0 (Figura 16.4a). Cuando el sistema vuelve a funcionar encuentra en el registro histórico el registro <T_0 iniciada>, pero no su correspondiente <T_0 comprometida> o <T_0 abortada>. Por tanto, la transacción T_0 debe deshacerse y se ejecutaría deshacer(T_0). Como resultado de esta operación, los saldos de las cuentas A y B (en el disco) se restituirían a 1.000 € y 2.000 €, respectivamente.

Suponga ahora que el fallo del sistema sucede justo después de haber escrito en almacenamiento estable el registro del registro histórico para el paso:

escribir(C)

de la transacción T_1 (Figura 16.4b). Cuando el sistema vuelva a funcionar, es necesario llevar a cabo dos acciones de recuperación. La operación deshacer(T_1) debe ejecutarse porque en el registro histórico aparece el registro <T_1 iniciada>, pero no aparecen <T_1 comprometida> ni <T_1 abortada>. La operación rehacer(T_0) debe ejecutarse porque el registro histórico contiene los registros <T_0 iniciada> y <T_0 comprometida>. Al final del procedimiento de recuperación, los saldos de las cuentas A, B y C son de 950 €, 2.050 € y 700 €, respectivamente.

Por último, suponga que el fallo del sistema tiene lugar justo después de haber escrito:

<T_1 comprometida>

en almacenamiento estable el registro del registro histórico (Figura 16.4c). Cuando el sistema vuelve a funcionar, deben rehacerse tanto T_0 como T_1 ya que se encuentran en el registro histórico los registros <T_0 iniciada> y <T_0 comprometida>, así como los registros <T_1 iniciada> y <T_1 comprometida>. Los saldos de las cuentas A, B y C después de la ejecución de los procedimientos de recuperación rehacer(T_0) y rehacer(T_1) serán 950 €, 2.050 € y 600 €, respectivamente.

16.3.6. Puntos de revisión

Cuando ocurre un fallo en el sistema se debe consultar el registro histórico para determinar las transacciones que deben rehacerse y las que deben deshacerse. En principio es necesario recorrer completamente el registro histórico para hallar esta información. En este enfoque hay dos inconvenientes principales:

1. El proceso de búsqueda consume tiempo.
2. La mayoría de las transacciones que deben rehacerse de acuerdo con el algoritmo ya tienen escritas sus actualizaciones en la base de datos. Aunque el hecho de volver a ejecutar estas transacciones no produzca resultados erróneos, sí repercutirá en un aumento del tiempo de ejecución del proceso de recuperación.

Para reducir este tipo de sobrecarga se introducen los puntos de revisión.

Más adelante se describe un esquema sencillo de puntos de revisión que (a) no permite que se realice ninguna operación de actualización mientras se realiza la operación de puntos de revisión y (b) da salida a todos los bloques modificados en la memoria intermedia al disco cuando se realice la operación de puntos de revisión. Posteriormente se tratará cómo modificar los procedimientos de puntos de revisión y recuperación para conseguir mayor flexibilidad y relajar estos requisitos.

Los puntos de revisión se realizan como sigue:

1. Escritura en almacenamiento estable de todos los registros del registro histórico que residan en ese momento en la memoria principal.
2. Escritura en disco de todos los bloques de memoria intermedia que se hayan modificado.
3. Escritura en almacenamiento estable de un registro del registro histórico <revisión L>, donde L es una lista de transacciones activas en el momento del punto de revisión.

Mientras se lleva a cabo un punto de revisión no se permite que ninguna transacción realice acciones de actualización, tales como escribir en un bloque de memoria intermedia o escribir un registro del registro histórico. Más adelante, en la Sección 16.5.2, se verá cómo se puede forzar este requisito.

La presencia de un registro <revisión L> en el registro histórico permite que el sistema pueda hacer más eficiente su procedimiento de recuperación. Considere una transacción T_i que se comprometió antes del punto de revisión. Para esa transacción el registro <T_i comprometida> (o el registro <T_i abortada>) aparece en el registro histórico antes que el registro <revisión>. Todas las modificaciones sobre la base de datos hechas por T_i se deben haber escrito en la base de datos antes del punto de revisión o formando parte del propio punto de revisión. Así, en el momento de la recuperación, no es necesario ejecutar una operación rehacer sobre T_i.

Cuando se produce un fallo, el esquema de recuperación examina el registro histórico para determinar el último registro <revisión L> (se puede hacer leyendo el registro desde el final hacia atrás, hasta que se encuentre un registro <revisión L>).

Las operaciones rehacer y deshacer han de ser aplicadas solo a las transacciones de L, y a todas las transacciones que iniciasen su ejecución después de que se escribiese el registro <revisión L> en el registro histórico. Sea este conjunto de transacciones T.

- Ejecutar deshacer(T_k) para todas las transacciones T_k de T para las que no exista un registro <T_k comprometida> o <T_k abortada> en el registro histórico.
- Ejecutar rehacer(T_k) para todas las transacciones T_k de T para las que aparezca un registro <T_k comprometida> o <T_k abortada> en el registro histórico.

Tenga en cuenta que solo hay que examinar la parte del registro histórico con el último registro de revisión para encontrar todas las transacciones de T, y para descubrir si existe un registro comprometida o abortada en dicho registro para cada transacción T.

A modo de ejemplo, considere el conjunto de transacciones {T_0, T_1, …, T_{100}}. Suponga que el último punto de revisión tiene lugar durante la ejecución de la transacción T_{67} y T_{69}, mientras que T_{68} y todas las transacciones con un subíndice inferior a 67 se han completado antes del punto de revisión. Así, durante el esquema de recuperación solo deben considerarse las transacciones T_{67}, T_{69}, …, T_{100}. Será necesario rehacer todas ellas si estas se comprometieron o abortaron, y será necesario deshacerlas en caso contrario, ya que están incompletas.

Considere el conjunto de transacciones L en un punto de revisión del registro histórico. Para cada transacción T_i de L se necesitan registros anteriores al punto de revisión para realizar la operación deshacer de la transacción, en el caso de que no comprometa. Sin embargo, una vez que se ha completado el punto de revisión no se necesitan los registros anteriores al primero de los registros <T_i iniciada>, entre las transacciones T_i de L. Estos registros históricos se pueden eliminar siempre que el sistema de bases de datos necesite el espacio que ocupan.

El requisito de que las transacciones no puedan realizar ninguna actualización en los bloques de memoria intermedia o en el registro histórico durante la operación de puntos de revisión puede ser engorroso, ya que el procesamiento de transacciones ha de detenerse mientras se esté realizando dicha operación. Un **punto de revisión difuso** es un punto de revisión en el que se permiten transacciones incluso mientras se escriben de salida los bloques de memoria intermedia. En la Sección 16.5.4 se describen los esquemas de puntos de revisión difusos. Más adelante, en la Sección 16.8, se describe un esquema de puntos de revisión que no solo es difuso sino que ni siquiera requiere que se escriban de salida los bloques de memoria intermedia al disco durante la operación de puntos de revisión.

16.4. Algoritmo de recuperación

Hasta este momento, sobre la recuperación se han identificado las transacciones necesarias para la operación rehacer y aquellas que se necesitan deshacer, pero no se ha indicado ningún algoritmo preciso para realizar estas acciones. Ya estamos en condiciones de presentar el algoritmo completo usando los registros del registro histórico para la recuperación de un fallo de transacciones y una combinación del último punto de revisión y el registro histórico para realizar la recuperación de un fallo del sistema.

El algoritmo de recuperación que se describe en esta sección requiere que no se puedan modificar los datos de las transacciones que no estén comprometidas por ninguna otra transacción, hasta que la primera esté comprometida o abortada. Recuerde que esta restricción ya se trató anteriormente, en la Sección 16.3.3.

16.4.1. Retroceso de transacciones

En primer lugar, considere el retroceso de las transacciones durante la operación normal (es decir, no durante la recuperación de un fallo del sistema). El retroceso de una transacción T_i se realiza como sigue:

1. Se explora el registro histórico desde atrás, y para cada registro T_i de la forma <T_i, X_j, V_1, V_2> que se encuentre:
   a. Se escribe el valor V_1 en el elemento de datos X_j.
   b. Se escribe un registro especial solo-rehacer <T_i, X_j, V_1> en el registro histórico, donde V_1 es el valor que se restablece en el elemento de datos X_j durante el retroceso. A estos registros se les denomina a veces **registros de compensación de registro histórico.** Estos registros no necesitan información de deshacer, ya que nunca se precisa deshacer una operación de deshacer. Se explicará más adelante cómo se usan.
2. Una vez que se encuentra <T_i iniciada> se detiene la exploración hacia atrás, y se escribe el registro <T_i abortada> en el registro histórico.

Observe que todas las acciones de actualización que realiza la transacción o en lugar de la transacción, incluyendo las acciones para la recuperación de elementos de datos a su valor anterior, no se han registrado en el registro histórico. En la Sección 16.4.2 se verá por qué es una buena idea.

16.4.2. Recuperación tras un fallo del sistema

Las acciones de recuperación cuando se vuelve a iniciar el sistema de base de datos después de un fallo se realizan en dos fases:

1. En la **fase rehacer** se vuelven a realizar modificaciones de *todas* las transacciones mediante la exploración hacia delante del registro histórico a partir del último punto de revisión. Los registros del registro histórico que se vuelven a ejecutar incluyen los registros de transacciones que retrocedieron antes de la caída del sistema y los que no se habían comprometido cuando ocurrió la caída.

Esta fase también determina todas las transacciones que estaban incompletas en el momento del fallo del sistema y, por tanto, se deben retroceder. Estas transacciones incompletas podrían haber estado activas en el momento del punto de revisión y aparecerían en la lista de transacciones del punto de revisión o podrían haberse iniciado posteriormente; además, estas transacciones incompletas no tendrían ni un <T_i abortada> ni <T_i comprometida> en el registro histórico.

Los pasos concretos que se siguen mientras se explora el registro histórico son los siguientes:

a. La lista de transacciones que hay que retroceder, la lista-deshacer, se establece inicialmente a la lista L del registro <revisión L> del registro histórico.

b. Cuando se encuentra un registro normal de la forma <T_i, X_j, V_1, V_2> o un registro de solo-rehacer de la forma <T_i, X_j, V_2>, la operación se rehace; es decir, se escribe el valor de V_2 en el elemento de datos X_j.

c. Cuando se encuentra un registro de la forma <T_i iniciada>, se añade T_i a la lista-deshacer.

d. Cuando se encuentra un registro de la forma <T_i abortada> o <T_i comprometida>, se elimina T_i de la lista-deshacer.

Al final de la fase rehacer, la lista-deshacer contiene todas las transacciones que están incompletas, esto es, ni comprometidas ni completado su retroceso antes del fallo del sistema.

2. En la **fase deshacer** retroceden todas las transacciones de la lista-deshacer. El retroceso se realiza recorriendo el registro histórico hacia atrás empezando por el final.

a. Cuando se encuentra un registro del registro histórico perteneciente a una transacción de la lista-deshacer se realizan las operaciones para deshacer de la misma manera que si el registro se hubiera encontrado durante el retroceso de una transacción fallida.

b. Cuando se encuentra en el registro histórico un registro <T_i iniciada> para una transacción T_i de la lista-deshacer, se escribe en el registro histórico un registro <T_i abortada> y elimina T_i de la lista-deshacer.

c. La fase deshacer termina cuando la lista-deshacer se vacía, es decir, el sistema ha encontrado registros <T_i iniciada> para todas las transacciones de la lista-deshacer.

Después de que termina la fase deshacer de recuperación, puede reanudarse el procesamiento normal de transacciones.

Observe que en la fase rehacer se vuelve a ejecutar cada registro del registro histórico desde que tuvo lugar el último punto de revisión. En otras palabras, esta fase de la recuperación al reiniciar repite todas las acciones de modificación que fueron ejecutadas después del punto de revisión y cuyos registros alcanzaron un registro histórico estable. Se incluyen aquí las acciones de transacciones incompletas y las acciones llevadas a cabo para retroceder transacciones fallidas. Las acciones se repiten en el mismo orden en el que se llevaron a cabo; de ahí que este proceso se denomine **repetición de la historia.** Aunque pueda parecer una pérdida de tiempo, al repetir la historia incluso para las transacciones fallidas, se simplifican los esquemas de recuperación.

En la Figura 16.5 se muestra un ejemplo de acciones que se registran durante el funcionamiento normal, y las acciones que se realizan durante la recuperación de reinicio. En el registro que se muestra en la figura, la transacción T_1 está comprometida, y la transacción T_0 se ha retrocedido totalmente antes del fallo del sistema. Observe cómo se recupera el valor del elemento de datos B durante el retroceso de T_0. Observe también el registro del punto de revisión con la lista de transacciones activas que contiene a T_0 y a T_1.

Cuando se recupera de un fallo del sistema, en la fase rehacer, el sistema aplica rehacer a todas las operaciones tras el último registro de revisión. En esta fase, la lista-deshacer contiene inicialmente T_0 y T_1; T_1 se elimina en primer lugar cuando se encuentra su registro de comprometida, mientras que se añade T_2 cuando se encuentra su registro de iniciada. La transacción T_0 se elimina de la lista-deshacer cuando se encuentra su registro de abortada, dejando en la lista-deshacer solo a T_2. La fase deshacer examina el registro hacia atrás desde el final, y cuando encuentra un registro de T_2 que actualiza A, se recupera el valor anterior de A, y se inscribe el registro de solo-rehacer en el registro histórico. Cuando se encuentra el registro de iniciada para T_2 se añade un registro abortada para T_2. Como la lista-deshacer no contiene más transacciones, la fase deshacer termina, acabando la recuperación.

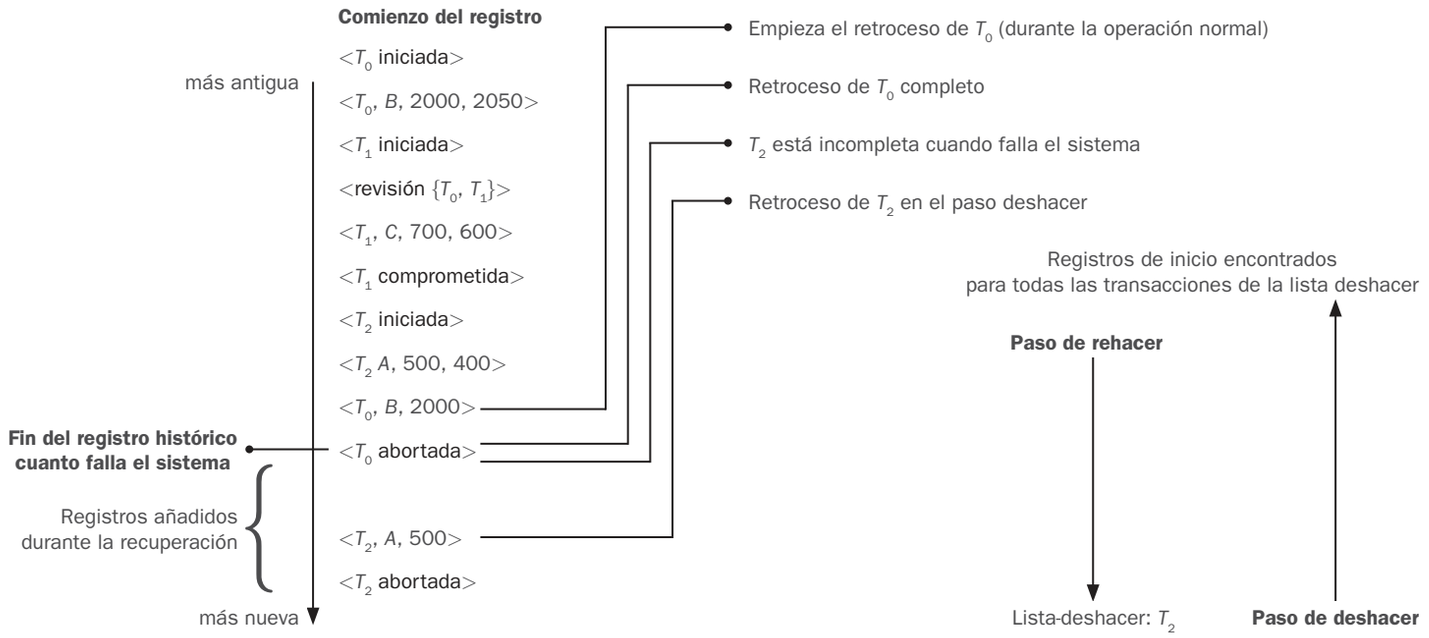


Figura 16.5. Ejemplo de acciones de registro y acciones durante la recuperación.

16.5. Gestión de memoria intermedia

En esta sección se consideran varios detalles sutiles que son esenciales para la implementación de un esquema de recuperación que garantice la consistencia de los datos y que lleve asociada una sobrecarga mínima respecto a la interacción con la base de datos.

16.5.1. Registro histórico con memoria intermedia

Hasta este momento se ha supuesto que cada registro del registro histórico se escribe en almacenamiento estable en el momento en que se crea. Esta suposición impone una sobrecarga muy alta en la ejecución del sistema por las siguientes razones: habitualmente, la unidad de escritura en almacenamiento estable es el bloque. En la mayoría de los casos un registro del registro histórico es mucho más pequeño que un bloque. Así, la escritura de cada registro del registro histórico se traduce en una escritura mucho mayor en el nivel físico. Además, como se vio en la Sección 16.2.1, la escritura de un bloque en almacenamiento estable puede suponer varias operaciones de escritura en el nivel físico.

El coste de realizar la escritura en almacenamiento estable de un bloque es suficientemente elevado como para que sea preferible escribir de una sola vez varios registros del registro histórico. Para ello, se escriben los registros del registro histórico en una memoria intermedia almacenada en la memoria principal en la que permanecen durante un tiempo hasta que se guardan en almacenamiento estable. Se pueden acumular varios registros del registro histórico en la memoria intermedia del registro histórico y escribirse en almacenamiento estable con una sola operación. El orden de los registros del registro histórico en el almacenamiento estable debe ser exactamente el mismo orden en el que fueron escritos en la memoria intermedia del registro histórico.

Debido a la utilización de la memoria intermedia del registro histórico, un registro del registro histórico puede permanecer únicamente en la memoria principal (almacenamiento volátil) durante un espacio de tiempo considerable. Como esos registros se perderían si hubiese un fallo del sistema, es necesario imponer nuevos requisitos sobre las técnicas de recuperación para garantizar la atomicidad de las transacciones:

- La transacción T_i pasa al estado comprometida después de que se haya escrito en almacenamiento estable el registro $<T_i$ comprometida$>$.
- Antes de escribir en almacenamiento estable el registro $<T_i$ comprometida$>$, todos los registros del registro histórico pertenecientes a la transacción T_i se deben escribir en almacenamiento estable.
- Antes de que un bloque de datos en memoria principal se pueda escribir en la base de datos (en almacenamiento no volátil) se deben haber escrito en almacenamiento estable todos los registros del registro histórico pertenecientes a los datos de ese bloque.

  Este último requisito se denomina regla de **registro de escritura anticipada (REA).** Estrictamente hablando, la regla REA solo necesita que haya sido puesta en almacenamiento estable la información concerniente a la operación deshacer y permite que la información relativa a la operación rehacer se escriba más tarde. La diferencia es relevante en aquellos sistemas en los que la información para rehacer y deshacer se guarda en registros del registro histórico independientes.

Las tres reglas representan situaciones en las que ciertos registros del registro histórico *deben* haber sido escritos en almacenamiento estable. No se produce ningún problema como resultado de la escritura de los registros del registro histórico *antes* de que sea necesaria. Así, cuando el sistema decide que es necesario escribir en almacenamiento estable un registro del registro histórico, puede escribir un bloque entero de ellos si hay suficientes registros en la memoria principal como para llenar un bloque. Si no hay suficientes registros para llenar el bloque, se forma un bloque parcialmente lleno con todos los registros que hubiera en la memoria principal y se ponen en almacenamiento estable.

La escritura en disco de la memoria intermedia del registro histórico a veces se denomina **forzar el registro histórico.**

16.5.2. Base de datos con memoria intermedia

En la Sección 16.2.2 se describió el uso de una jerarquía de almacenamiento de dos niveles. La base de datos se almacena en almacenamiento no volátil (disco) y, cuando sea necesario, se traen a la memoria principal los bloques de datos que hagan falta. Como la memoria principal suele ser mucho más pequeña que la base de datos completa, puede ser necesaria la sobrescritura de un bloque B_1 en la memoria principal cuando sea necesario traer a la memoria otro bloque B_2. Si B_1 ha sido modificado, se debe escribir B_1 antes de traer B_2. Como se estudió en la Sección 16.8.1, en el Capítulo 10, esta jerarquía de almacenamiento se corresponde con el concepto usual de *memoria virtual*.

Se podría esperar que las transacciones forzasen la salida de todos los bloques modificados al disco cuando se comprometen. Este enfoque se denomina política de **forzar.** La alternativa, la política de **no-forzar,** permite que se comprometa la transacción incluso aunque haya modificado algunos bloques y no se hayan escrito todavía en el disco. Todos los algoritmos de recuperación que se describen en este capítulo funcionan correctamente incluso con una política de no-forzar. Esta política de no-forzar permite un compromiso más rápido de las transacciones; más aún, permite que se acumulen varias actualizaciones en un bloque antes de que se les dé salida a almacenamiento estable, lo que reduce en gran medida el número de operaciones de salida para bloques que se actualizan con mucha frecuencia. En este sentido, el enfoque habitual que suelen tomar la mayoría de los sistemas es de no-forzar.

De forma similar, se podría esperar que no se escriban en disco los bloques que modifican una transacción que continúa activa. Esta política se denomina de **no-apropiación.** La alternativa, la política de **apropiación,** permite al sistema que escriba en disco bloques modificados incluso aunque la transacción que hizo las modificaciones no haya pasado a comprometida. Siempre que se siga la regla de registro de escritura anticipada, todos los algoritmos de recuperación que se tratan en este capítulo funcionan correctamente incluso con una directiva de apropiación. Más aún, la política de no-apropiación no funciona con transacciones que realizan un gran número de actualizaciones, ya que la memoria intermedia se puede llenar de páginas actualizadas que no se pueden sacar al disco, y las transacciones no pueden continuar. Por ello, el enfoque habitual de la mayoría de los sistemas es utilizar la política de apropiación.

Para ilustrar la necesidad del requisito de registro histórico con escritura anticipada considere el ejemplo bancario con las transacciones T_0 y T_1. Suponga que el estado del registro histórico es:

$$<T_0 \text{ iniciada}>$$
$$<T_0, A, 1000, 950>$$

y que la transacción T_0 realiza una operación leer(B). Suponga también que el bloque en el que se encuentra B no está en memoria

principal y que esa memoria principal está llena. Suponga que el bloque en el que reside A es el elegido para escribirse en disco. Si el sistema escribe este bloque en disco y luego el sistema falla, los valores para las cuentas A, B y C en la base de datos son 950 €, 2.000 € y 700 €, respectivamente. Este estado de la base de datos es inconsistente. Sin embargo, según los requisitos de la regla REA, el registro del registro histórico:

$$<T_0, A, 1000, 950>$$

debe escribirse en almacenamiento estable antes de producirse la escritura del bloque en el que se encuentra A. El sistema puede usar ese registro del registro histórico durante la recuperación para devolver la base de datos a un estado consistente.

Cuando se va a escribir un bloque B_1 en el disco, todos los registros del registro histórico que pertenecen a los datos de B_1 se deben escribir en almacenamiento estable antes de que se escriba B_1. Es importante que no haya escrituras en marcha en el bloque B_1 mientras escribe de salida el bloque, ya que este tipo de escritura viola la regla de registro de escritura anticipada. Se puede asegurar que no hay escrituras en proceso usando un bloqueo especial:

- Antes de que una transacción realice una escritura de un elemento de datos, adquiere un bloqueo exclusivo del bloque en el que residen los elementos de datos. El bloqueo se libera inmediatamente después de realizarse la actualización.
- La secuencia de acciones que ha de llevar a cabo el sistema cuando se escribe un bloque de salida sería la siguiente:
  - Obtener un bloqueo exclusivo del bloque, para asegurarse de que no hay ninguna transacción escribiendo en el bloque.
  - Escritura en almacenamiento estable de los registros del registro histórico hasta que todos los registros pertenecientes al bloque B_1 se hayan escrito.
  - Escritura en disco del bloque B_1.
  - Liberar el bloqueo cuando la salida del bloque se haya completado.

Los bloqueos de los bloques en la memoria intermedia no están relacionados con los bloqueos que se usan para el control de concurrencia de las transacciones y al liberarlos en una forma que no sea de dos fases no tiene ninguna implicación sobre la secuencialidad de la transacción. Estos bloqueos, y otros similares que se mantienen durante un muy corto espacio de tiempo, se suelen llamar **pestillos.**

Los bloqueos de los bloques de memoria intermedia se usan también para asegurar que los bloques de la memoria intermedia no se actualizan y no se generan los registros del registro histórico, mientras se realiza un punto de revisión. Esta restricción se puede forzar adquiriendo un bloqueo exclusivo de todos los bloques de memoria intermedia, así como un bloqueo exclusivo en el registro histórico, antes de realizar la operación de punto de revisión. Estos bloqueos se pueden liberar en cuanto sea posible cuando se complete la operación de punto de revisión.

Los sistemas de bases de datos suelen tener un proceso que continuamente explora los bloques de memoria intermedia dando salida a los bloques modificados hacia el disco. El protocolo de bloqueo anterior hay que seguirlo cuando se da salida a los bloques. Entonces, por la salida continua de bloques modificados, el número de **bloques sucios** en la memoria intermedia, es decir, bloques modificados en la memoria intermedia pero que no se han escrito de salida, se minimiza. Por tanto, el número de bloques a los que hay que dar salida durante un punto de revisión se minimiza; más aún, cuando se necesita expulsar a un bloque de la memoria intermedia es muy probable que exista un bloque no sucio para su expulsión, lo que permite leer y dar entrada inmediatamente, en lugar de tener que esperar a que se complete una salida.

16.5.3. La función del sistema operativo en la gestión de la memoria intermedia

La memoria intermedia de la base de datos puede gestionarse usando uno de estos dos enfoques:

1. El sistema de base de datos reserva parte de la memoria principal para utilizarla como memoria intermedia y es él, en vez del sistema operativo, el que se encarga de gestionarlo. El sistema de base de datos gestiona la transferencia de los bloques de datos de acuerdo con los requisitos de la Sección 16.5.2.

   El inconveniente de este enfoque es que limita la flexibilidad en la utilización de la memoria principal. El tamaño de la memoria intermedia no debe ser muy grande para que otras aplicaciones tengan suficiente espacio disponible en la memoria principal para sus propias necesidades. Sin embargo, incluso cuando ninguna otra aplicación esté en ejecución, la base de datos no podrá hacer uso de toda la memoria disponible. Asimismo, aquellas aplicaciones que no tienen nada que ver con la base de datos no pueden usar la región de la memoria reservada para la memoria intermedia de la base de datos aunque no se estén utilizando algunas de las páginas almacenadas en la memoria intermedia.

2. El sistema de base de datos implementa su memoria intermedia dentro de la memoria virtual del sistema operativo. Como el sistema operativo conoce los requisitos de memoria de todos los procesos del sistema, es lógico que pueda decidir los bloques de la memoria intermedia que deben escribirse en el disco y el momento en el que debe realizarse esta escritura. Pero para garantizar los requisitos del registro histórico de escritura anticipada de la Sección 16.5.1, el sistema operativo no debería realizar él mismo la escritura de las páginas de la base de datos de la memoria intermedia, sino que debería pedírselo al sistema de base de datos para que fuera este el que forzara la escritura de los bloques de la memoria intermedia.

   Lamentablemente, casi todos los sistemas operativos actuales ejercen un control completo sobre la memoria virtual. El sistema operativo reserva espacio en el disco para almacenar las páginas de memoria virtual que no se encuentran en ese momento en la memoria principal; este espacio se denomina **espacio de intercambio.** Si el sistema operativo decide escribir un bloque B_x, ese bloque se escribe en el espacio de intercambio del disco, por lo que el sistema de base de datos no tiene forma de controlar la escritura de los bloques de la memoria intermedia.

   Por consiguiente, si la memoria intermedia de la base de datos está en la memoria virtual, las transferencias entre los archivos de la base de datos y la memoria intermedia en memoria virtual deben estar gestionadas por el sistema de base de datos, hecho que subraya el cumplimiento de los requisitos del registro histórico de escritura anticipada visto anteriormente.

   Este enfoque puede provocar una escritura adicional de datos en el disco. Si el sistema operativo realiza la escritura de un bloque B_x, este no se escribe en la base de datos sino que se escribe en el espacio de intercambio que utiliza la memoria virtual del sistema operativo. Cuando la base de datos necesita escribir B_x, el sistema operativo puede necesitar primero leer B_x de su espacio de intercambio. Así, en lugar de realizar una sola escritura de B_x, son necesarias dos escrituras (una del sistema operativo y otra del sistema de base de datos), así como una lectura adicional.

Aunque ambos enfoques tienen algunos inconvenientes, debe elegirse cualquiera de los dos, excepto si el sistema operativo está diseñado para soportar los requisitos del registro histórico de base de datos.

16.5.4. Puntos de revisión difusos

La técnica de puntos de revisión descrita en la Sección 16.3.6 requiere que, mientras se efectúa el punto de revisión, se suspendan temporalmente todas las modificaciones de la base de datos. Si el número de páginas de la memoria intermedia es grande, un punto de revisión puede llevar mucho tiempo, lo que puede provocar una interrupción inaceptable en el procesamiento de transacciones.

Para evitar estas interrupciones es posible modificar la técnica para permitir modificaciones después de haber escrito en el registro histórico el registro revisión, pero antes de escribir en disco los bloques de la memoria intermedia que han sufrido modificaciones. El punto de revisión así generado recibe el nombre de **punto de revisión difuso.**

Dado que las páginas se escriben en disco solo después de que se haya escrito el registro revisión, es posible que se produzca un fallo del sistema antes de que todas las páginas se hayan escrito. Por tanto, los puntos de revisión en disco pueden estar incompletos. Una forma de manejar los puntos de revisión incompletos es la siguiente: la ubicación del registro de revisión en el registro histórico del último punto de revisión completado se almacena en una posición fijada, última-revisión, en disco. El sistema no actualiza esta información cuando escribe el registro revisión. En su lugar, antes de escribirlo, crea una lista de todos los bloques de memoria intermedia modificados. La información de última-revisión solo se actualiza cuando todos los bloques de memoria intermedia de la lista de bloques modificados se hayan escrito en disco.

Incluso con la revisión difusa, no se debe actualizar un bloque de memoria intermedia mientras se esté escribiendo en disco, aunque sí se pueden actualizar concurrentemente otros bloques. Una vez que se han escrito en disco todos los bloques, debe seguirse el protocolo de registro histórico de escritura anticipada.

16.6. Fallo con pérdida de almacenamiento no volátil

Hasta ahora solo se ha considerado el caso en el que un fallo conduce a la pérdida de información residente en almacenamiento volátil mientras que el contenido del almacenamiento no volátil permanecía intacto. A pesar de que es raro encontrarse con un fallo en el que se pierda información de almacenamiento no volátil, es necesario prepararse para afrontar este tipo de fallos. En esta sección solo se hablará del almacenamiento en disco. La argumentación puede aplicarse también a otras clases de almacenamiento no volátil.

La idea básica es **volcar** periódicamente (digamos, una vez al día) el contenido entero de la base de datos en almacenamiento estable. Por ejemplo, puede volcarse la base de datos en una o más cintas magnéticas. Se utilizará el volcado más reciente para que la base de datos recupere un estado consistente cuando ocurra un fallo que conduzca a la pérdida de algunos bloques físicos de la base de datos. Una vez que se complete esta operación, el sistema utilizará el registro histórico para llevar al sistema de base de datos al último estado consistente en el que estuvo antes de producirse el fallo.

Un enfoque sobre el volcado de la base de datos requiere que ninguna transacción pueda estar activa durante el procedimiento de volcado y utiliza un procedimiento similar al de los puntos de revisión:

1. Escribir en almacenamiento estable todos los registros del registro histórico que residan en ese momento en la memoria principal.
2. Escribir en disco todos los bloques de la memoria intermedia.
3. Copiar el contenido de la base de datos en almacenamiento estable.
4. Escribir el registro del registro histórico <volcar> en almacenamiento estable.

Los pasos 1, 2 y 4 se corresponden con los tres pasos utilizados para realizar un punto de revisión en la Sección 16.3.6.

Para la recuperación por pérdida de almacenamiento no volátil se restituye la base de datos en el disco utilizando el último volcado realizado. Entonces se consulta el registro histórico y se rehacen todas las transacciones que se hubieran comprometido desde que se efectuó ese último volcado. Observe que no es necesario ejecutar ninguna operación deshacer.

En el caso de que se produzca un fallo parcial del almacenamiento no volátil, como es el fallo de un único bloque o unos pocos bloques, solo hay que restaurar esos pocos bloques y rehacer solo las acciones realizadas en esos bloques.

Un volcado del contenido de una base de datos se denomina también **volcado de archivo,** ya que pueden archivarse los volcados y utilizarlos más tarde para examinar estados anteriores de la base de datos. Los volcados de una base de datos y la realización de puntos de revisión de las memorias intermedias son dos procesos análogos.

La mayoría de los sistemas de bases de datos disponen también de un **volcado SQL,** que escribe sentencias SQL DDL y sentencias insertar de SQL en un archivo, que se pueden después volver a ejecutar para reconstruir la base de datos. Este volcado resulta útil cuando se migran datos a un ejemplar diferente de la base de datos o a una versión diferente del software de bases de datos, ya que las ubicaciones físicas y la configuración pueden diferir en el otro ejemplar de la base de datos o la otra versión del software.

El procedimiento de volcado simple que se ha descrito anteriormente es costoso debido a las dos razones siguientes. En primer lugar, debe copiarse en almacenamiento estable la base de datos entera, lo que conlleva una considerable transferencia de datos. En segundo lugar, se pierden ciclos de CPU porque se detiene el procesamiento de transacciones durante el procedimiento de volcado. Se han desarrollado esquemas de **volcado difuso** que permiten que las transacciones sigan activas mientras se realiza el volcado. Son esquemas similares a los de los puntos de revisión difusos. Para más detalles consulte las notas bibliográficas.

16.7. Liberación rápida de bloqueos y operaciones de deshacer lógicas

Cualquier índice que se use en el procesamiento de una transacción, como un árbol B^+, se puede tratar como datos normales, pero para incrementar la concurrencia se puede utilizar el algoritmo de control de concurrencia descrito en la Sección 15.10 para permitir liberar pronto los bloqueos, en forma que no sea en dos fases. El resultado de esta liberación rápida de bloqueos es que es posible que un valor de un nodo del árbol B^+ lo actualice una transacción T_1, insertando una entrada (*V1*, *R1*), y después, otra transacción T_2 inserte una entrada (*V2*, *R2*) en el mismo nodo, desplazando la entrada (*V1*, *R1*) incluso antes de que T_1 complete su ejecución.[4] En este momento, no se puede deshacer la transacción T_1 sustituyendo el contenido del nodo por el valor anterior a que T_1 realizase la inserción, ya que también desharía la inserción que ha realizado T_2; la transacción T_2 todavía debe pasar a comprometida (o puede que ya lo haya hecho). En este ejemplo, la única forma de deshacer el efecto de insertar (*V1*, *R1*) es ejecutar una operación de borrado.

En el resto de esta sección se verá cómo extender el algoritmo de recuperación de la Sección 16.4 para disponer de bloqueos de liberación rápida.

4 Recuerde que una entrada consta de un valor de clave y un identificador de registro, o un valor de clave y un registro en el caso de un nivel de hoja de una organización de archivo de árbol B^+.

16.7.1. Operaciones lógicas

Las operaciones de inserción y borrado son ejemplos de un tipo de operaciones que requieren operaciones de deshacer lógicas ya que realizan una liberación rápida de los bloqueos; a estas operaciones se les llama **operaciones lógicas.** La liberación rápida de los bloqueos no es solo importante para los índices, sino también para otras operaciones en las estructuras de datos del sistema que se actualizan con mucha frecuencia; algunos ejemplos incluyen las estructuras de datos que realizan el seguimiento de los registros que contienen bloques de una relación, el espacio libre de un bloque y los bloques libres de la base de datos. Si no se realiza una liberación rápida de los bloqueos tras realizar una operación en estas estructuras de datos, las transacciones tienden a ejecutarse secuencialmente, lo que afecta al rendimiento del sistema.

La teoría de la secuencialidad de los conflictos se ha extendido a las operaciones, según qué operaciones entran en conflicto con otras. Por ejemplo, dos operaciones de inserción en un árbol B^+ no entran en conflicto si insertan valores de clave diferentes, incluso aunque realicen la inserción en áreas que se superponen de la misma página de índices. Sin embargo, las operaciones de inserción y borrado entran en conflicto con otras operaciones de inserción y borrado, así como las operaciones de lectura, si utilizan los mismos valores de clave. Consulte las notas bibliográficas para más información sobre este tema.

Las operaciones adquieren *bloqueos de bajo nivel* mientras se ejecutan, y los liberan cuando terminan; sin embargo, la transacción correspondiente debe mantener el *bloqueo de alto nivel* en la forma de dos fases para evitar que otras transacciones concurrentes ejecuten operaciones que entren en conflicto. Por ejemplo, cuando se realiza una inserción en una página de un árbol B^+, se adquiere un bloqueo de corto plazo sobre la página, lo que permite desplazar las entradas de la página durante la inserción; el bloqueo de corto plazo se libera en cuanto la página se actualiza. Esta liberación del bloqueo de corto plazo permite que se ejecute otra inserción en la misma página. Sin embargo, cada transacción debe obtener un bloqueo sobre las claves que se insertan o borran, y mantenerlo en la forma de dos fases, para evitar que una transacción concurrente ejecute un operación de inserción, borrado o lectura sobre el mismo valor de clave.

Una vez se libera el bloqueo de bajo nivel, la operación no se puede deshacer utilizando los valores anteriores de los elementos de datos actualizados y, en su lugar, se debe deshacer ejecutando una operación de compensación; esta operación se llama **operación de deshacer lógica.** Es importante que los bloqueos de bajo nivel que se adquieren durante la operación sean suficientes para realizar un deshacer lógico de la operación, por razones que se explican en la Sección 16.7.4.

16.7.2. Registros de deshacer lógicos en el registro histórico

Para disponer de operaciones de deshacer lógicas, antes de que la operación realice la modificación de un índice, la transacción crea un registro $<T_i, O_j,$ inicio-operación$>$ en el registro histórico, donde O_j es un identificador único para la operación.[5] Mientras el sistema ejecuta la operación, crea un registro de actualización en el registro histórico de la forma normal para las actualizaciones de dicha operación. Por tanto, se escriben los valores anterior y nuevo de la forma normal para cada actualización que realiza la operación; es necesario el valor anterior en el caso de que se necesite retroceder la operación antes de que se complete. Cuando la operación termina, escribe un registro fin-operación en el registro histórico de la forma $<T_i, O_j,$ fin-operación, $U>$, donde U indica la información de deshacer.

Por ejemplo, si la operación insertó una entrada en un árbol B^+, la información de deshace U debería indicar que se debe realizar una operación de borrado y debería identificar el árbol B^+ y qué entrada del árbol eliminar. Este registro de información sobre las operaciones se denomina **registro histórico lógico.** En cambio, el registro histórico de la información anterior y nueva se denomina **registro histórico físico** y el registro correspondiente al registro histórico se denomina **registro del registro histórico físico.**

Tenga en cuenta que en el esquema anterior, el registro histórico lógico solo se usa para operaciones de deshacer y no para rehacer; las operaciones de rehacer se realizan usando solamente los registros del registro histórico físico. Es así porque el estado de la base de datos tras un fallo del sistema puede reflejar algunas actualizaciones de una operación pero no otras, dependiendo de si los bloques en memoria intermedia se han escrito en disco o no antes del fallo del sistema. Las estructuras como los árboles B^+ no estarán en un estado consistente, y no se pueden realizar operaciones lógicas, ni tampoco rehacer o deshacer sobre una estructura de datos inconsistente. Para realizar operaciones lógicas rehacer o deshacer el estado de la base de datos en el disco debe ser **consistente en cuanto a operaciones,** es decir, no debería haber efectos parciales de ninguna operación. Sin embargo, como se verá más adelante, el procesamiento del registro histórico físico en la fase de rehacer del esquema de recuperación junto con el procesamiento de deshacer utilizando los registros del registro histórico físico asegura que las partes de la base de datos a las que se accede con una operación de deshacer lógica están en un estado consistente en cuanto a operaciones, antes de que se realice la operación de deshacer lógica.

Una operación se dice que es **idempotente** si cuando se ejecuta varias veces sobre una fila da el mismo resultado que si se ejecuta una vez. Las operaciones como las inserciones de una entrada en un árbol B^+ pueden ser no idempotentes, y los algoritmos de recuperación deben tener cuidado de si estas operaciones ya se han realizado para no volverlas a ejecutar. Por otra parte, un registro del registro físico es idempotente, ya que el elemento de datos correspondiente tendría el mismo valor independientemente de si la actualización del registro histórico se ejecuta una vez o varias.

16.7.3. Retroceso de transacciones con deshacer lógico

Cuando se hace retroceder una transacción T_i, se explora el registro histórico desde el final y se procesan los registros correspondientes a T_i de la siguiente forma:

1. Los registros del registro histórico físicos que se encuentren durante la exploración se manejan como se ha indicado anteriormente, excepto aquellos que se omiten, como se describirá en breve. Las operaciones lógicas incompletas de deshacer usando los registros que genera la operación.
2. Las operaciones lógicas completas, identificadas con los registros fin-operación, se retroceden de forma diferente. Cuando el sistema encuentra un registro $<T_i, O_j,$ fin-operación, $U>$, toma acciones especiales:
   a. Retrocede la operación usando la información de deshacer U del registro. Registra la actualización que realiza durante el retroceso de la operación, al igual que las actualizaciones lo hacen cuando se ejecutan por primera vez.

      Al final de la operación de retroceso, en lugar de generar un registro del registro histórico $<T_i, O_j,$ fin-operación, $U>$, el sistema de bases de datos genera $<T_i, O_j,$ abortar-operación$>$.

5 Se puede usar como identificador único la posición en el registro histórico del registro inicio-operación.

b. Cuando continúa el recorrido hacia atrás del registro histórico, el sistema omite todos los registros del registro histórico de la transacción hasta que encuentra el registro <T_i, O_j, inicio-operación>. Una vez encuentra el registro inicio-operación en el registro histórico, el resto de registros referentes a la transacción se procesa de nuevo normalmente.

Observe que el sistema registra información física de deshacer para las actualizaciones que se realizan durante el retroceso, en lugar de usar registros del registro histórico de compensación de solo-lectura. Se debe a que puede producirse un fallo del sistema mientras se está realizando un deshacer lógico, y durante la recuperación del sistema tiene que completar el deshacer lógico; para ello, al reiniciar la recuperación se desharán los efectos parciales del deshacer anterior, usando la información de deshacer física y después se hará de nuevo el deshacer lógico.

Observe también que, durante el retroceso, la omisión de registros del registro histórico físico cuando se encuentra el registro fin-operación asegura que no se usen para el retroceso valores anteriores del registro una vez terminada la operación.

3. Si el sistema encuentra un registro <T_i, O_j, abortar-operación>, omite todos los registros precedentes (incluyendo el registro fin-operación de O_j) hasta que se encuentra el registro <T_i, O_j, inicio-operación>.

Se encontraría un registro abortar-operación solo si una transacción que se estuviera retrocediendo hubiera sido parcialmente retrocedida antes. Recuerde que las operaciones lógicas puede que no sean idempotentes y, por tanto, una operación lógica de deshacer no se puede realizar varias veces. Estos registros precedentes se deben omitir para evitar un retroceso múltiple de la misma operación, en el caso de que haya ocurrido un fallo del sistema durante un retroceso anterior y la transacción haya retrocedido a medias.

4. Como anteriormente, cuando se encuentra el registro <T_i inicio>, el retroceso de la transacción termina y el sistema añade un registro <T_i abortada> al registro histórico.

Si sucede un fallo mientras se está ejecutando una operación lógica, no se encontrará el registro fin-operación de la operación cuando la transacción retroceda. Sin embargo, para cada actualización realizada por la operación hay disponible en el registro histórico información para deshacer (en la forma de registros con el valor anterior en el registro histórico físico). Los registros del registro histórico físico se usarán para hacer retroceder la operación incompleta.

Ahora suponga que se estaba realizando una operación de deshacer cuando ocurrió un fallo del sistema, que podría haber sucedido si una transacción se estaba retrocediendo cuando se produjo el fallo del sistema. Entonces el registro del registro histórico físico que se escribieron durante la operación de deshacer se encontrarían y la propia operación de deshacer parcial se desharía a sí misma usando estos registros físicos. Continuando con la exploración hacia atrás del registro histórico, se encontraría la operación fin-operación original y se volvería a ejecutar la operación deshacer de nuevo. Al retroceder los efectos parciales de una operación deshacer anterior utilizando los registros físicos lleva a la base de datos a un estado consistente, lo que permite que se ejecute de nuevo la operación de deshacer lógica.

En la Figura 16.6 se muestra un ejemplo de un registro histórico generado por dos transacciones, que añaden o sustraen un valor de un elemento de datos. La liberación rápida de bloqueos en el elemento de datos C por la transacción T_0 después de terminar la operación O_1 permite que la transacción T_1 actualice el elemento de datos usando O_2 incluso antes de que termine T_0, pero necesita un deshacer lógico. La operación deshacer lógica necesita añadir o sustraer un valor del elemento de datos, en lugar de restaurar un valor anterior al elemento de datos.

Las anotaciones en la figura indican que antes de que termine la operación, el retroceso puede realizar un deshacer físico; cuando la operación termina y libera los bloqueos de bajo nivel, la operación deshacer debe realizarse sustrayendo o añadiendo un valor, en lugar de restaurar el valor anterior. En el ejemplo de la figura, T_0 retrocede la operación O_1 añadiendo 100 a C; por otra parte, para el elemento de datos B que no fue objeto de una liberación rápida de bloqueos, deshacer se realiza físicamente. Observe que T_1, que ha realizado una actualización en C pasa a comprometida, y su actualización O_2 que añade 200 a C y se realiza antes de deshacer O_1, persiste incluso aunque O_1 se haya deshecho.

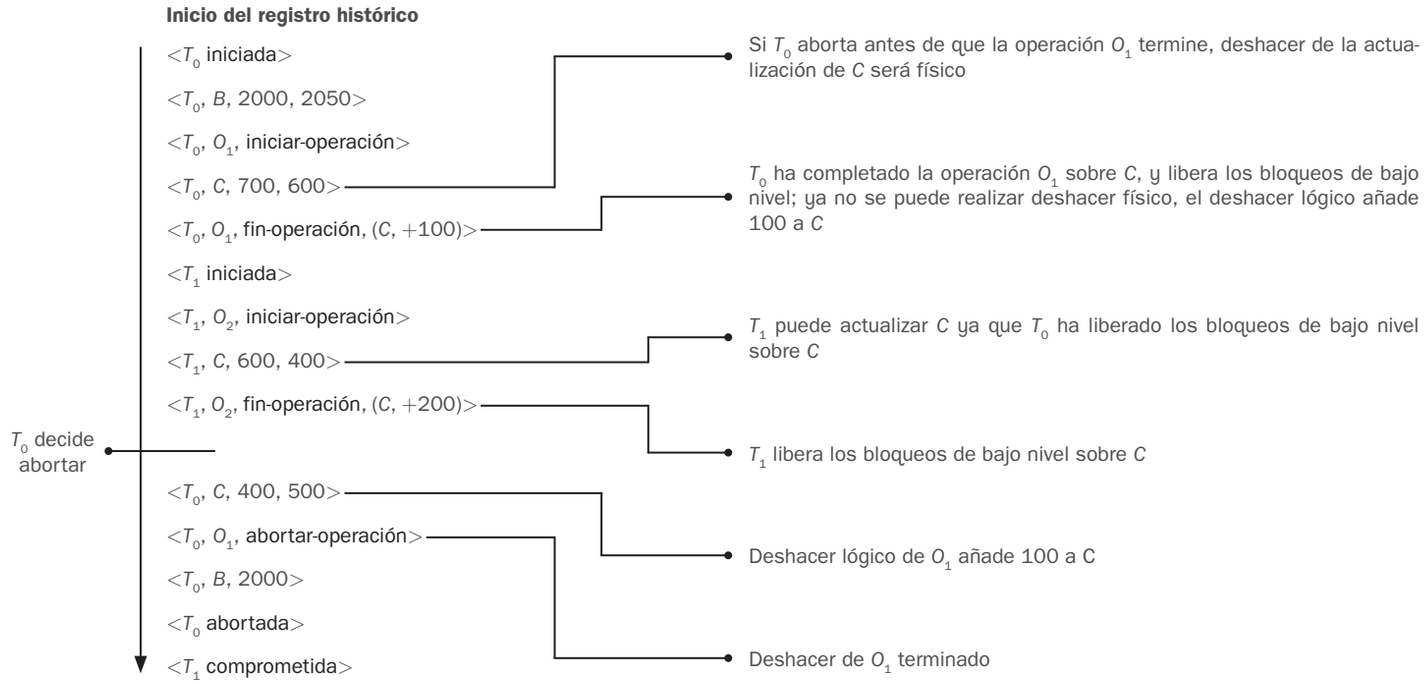


Figura 16.6. Retroceso de una transacción con operaciones de deshacer lógicas.

En la Figura 16.7 se muestra un ejemplo de recuperación de un fallo del sistema con un deshacer con registro histórico lógico. En este ejemplo, la operación T_1 estaba activa y ejecutando la operación en el momento del punto de revisión. En el paso de rehacer, las acciones de O_4 que están en el registro del registro histórico después del punto de revisión se rehacen. En el momento del fallo del sistema T_2 estaba ejecutando la operación O_5 pero la operación se había terminado. La lista de deshacer contiene a T_1 y a T_2 al final del paso de rehacer. Durante el paso de deshacer, se realiza la operación deshacer de la operación O_5 utilizando el valor anterior en el registro físico, estableciendo el valor de C a 400; esta operación se escribe en el registro histórico usando un registro de solo-rehacer. A continuación se encuentra el registro iniciada de T_2, por lo que se añade <T_2 abortada> al registro y se borra T_2 de la lista-deshacer.

El siguiente registro que se encuentra es el registro fin-operación de O_4; se realiza un deshacer lógico para esta operación añadiendo 300 a C, que se registra físicamente y se añade un registro en el registro histórico abortar-operación para O_4. Los registros físicos que formaban parte de O_4 se omiten hasta que se encuentra el registro iniciar-operación para O_4. En este ejemplo no existen otros registros implicados, pero en general se pueden encontrar registros de otras transacciones antes de que le llegue al registro iniciar-operación; por supuesto, estos registros no se omiten (a no ser que formen parte de una operación completa para la que la transacción correspondiente y el algoritmo omiten esos registros). Después de encontrar el registro iniciar-operación para O_4 se encuentra un registro físico para T_1, que se retrocede físicamente.

Finalmente, se encuentra un registro iniciada para T_1; entonces se añade un registro <T_1 abortada> al registro histórico y se borra <T_2 abortada> de la lista-deshacer. En este momento la lista-deshacer está vacía y se termina la fase de deshacer.

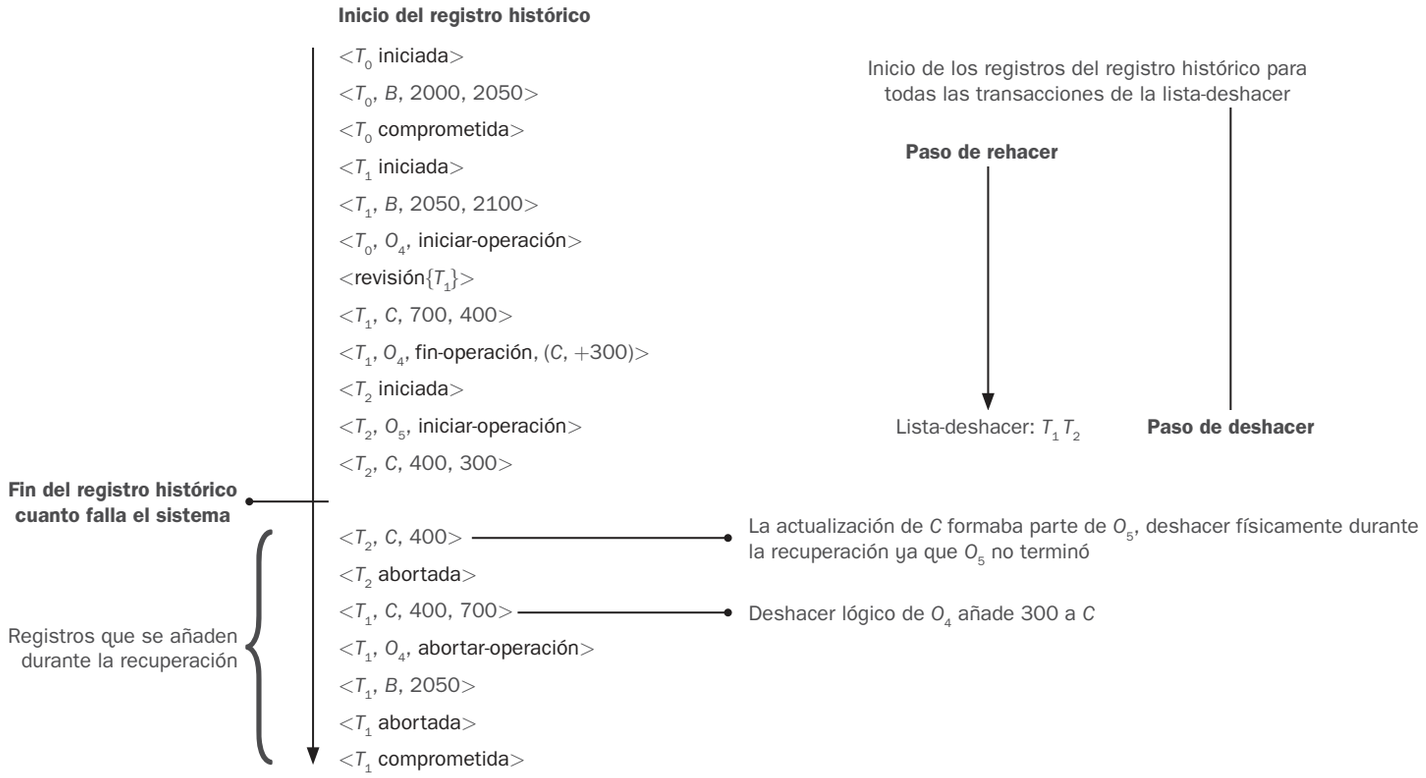


Figura 16.7. Acciones de recuperación de fallos con operaciones de deshacer lógicas.

16.7.4. Aspectos de concurrencia y deshacer lógico

Como se ha mencionado anteriormente, es importante que los bloqueos de bajo nivel que se adquieran durante una operación sean suficientes para realizar un deshacer lógico posterior de la operación; en caso contrario se pueden ejecutar operaciones concurrentes durante el procesamiento normal que causen problemas en la fase de deshacer. Por ejemplo, suponga el deshacer lógico de la operación O_1 para la transacción T_1 que puede entrar en conflicto en el nivel de elemento de datos con la operación O_2 de la transacción T_2 y O_1 termina mientras que O_2 no. Suponga también que ninguna de las transacciones ha pasado al estado de comprometida cuando el sistema falla. El registro físico del registro histórico de O_2 puede aparecer antes y después del registro fin-operación de O_1 y, durante la recuperación, las actualizaciones que se realizan durante el deshacer lógico de O_1 pueden sobrescribirse parcial o totalmente por los valores anteriores durante el deshacer físico de O_2. Este problema no ocurriría si O_1 hubiese obtenido todos los bloqueos de bajo nivel necesarios para el deshacer lógico de O_1, ya que entonces no habría tal concurrente O_2.

Si tanto la operación original como su operación de deshacer lógico acceden a una única página (tales operaciones se denominan operaciones fisiológicas y se tratarán en la Sección 16.8), el requisito de bloqueo anterior se cumple fácilmente. En caso contrario, se necesitan los detalles de la operación concreta para considerar cuándo decidir si se necesitan obtener los bloqueos de bajo nivel. Por ejemplo, las operaciones de actualización en un árbol B^+ podrían obtener un bloqueo de corto plazo en la raíz para asegurarse de que las operaciones se ejecutan secuencialmente. Consulte las notas bibliográficas sobre control de concurrencia en árboles B^+ y la recuperación utilizando los registros históricos de deshacer lógicos. Consulte las notas bibliográficas para más información sobre enfoques alternativos, llamados recuperación multinivel, que relaja los requisitos de bloqueo.

16.8. ARIES**

El método de recuperación ARIES es un representante de los métodos actuales de recuperación. La técnica de recuperación que se ha descrito en la Sección 16.4, junto con las técnicas de registro histórico de deshacer lógico que se han descrito en la Sección 16.7, se han modelado después de ARIES, pero se han simplificado significativamente para ilustrar los conceptos clave y hacerlas más fácil de comprender. En cambio, ARIES utiliza varias técnicas para reducir el tiempo de recuperación y la sobrecarga de los puntos de revisión. En particular, ARIES es capaz de evitar rehacer muchas operaciones registradas que ya se han realizado y de reducir la cantidad de información registrada. El precio pagado supone una mayor complejidad, pero los beneficios merecen la pena.

Las diferencias principales entre ARIES y el algoritmo de recuperación avanzada expuesto son que ARIES:

1. Usa un **número de secuencia del registro histórico (NSR)** para identificar a los registros del registro histórico, y guarda esos números en páginas de la base de datos para identificar las operaciones que se han realizado sobre una página de la base de datos.
2. Soporta operaciones **rehacer fisiológicas,** que son físicas en el sentido de que la página afectada está físicamente identificada, pero que pueden ser lógicas en la página.

   Por ejemplo, el borrado de un registro de una página puede provocar que muchos otros registros de la página se desplacen si se usa una estructura de páginas con ranuras (Sección 10.5.2). Con el registro histórico rehacer físico, hay que registrar todos los bytes de la página afectada por el desplazamiento de los registros. Con el registro histórico fisiológico, la operación borrado se puede registrar, lo que da lugar a un registro mucho más pequeño. Al rehacer la operación borrado se borraría el registro y se desplazarían los registros que hiciera falta.
3. Emplea una **tabla de páginas sucias** para minimizar las operaciones rehacer innecesarias durante la recuperación. Como ya se ha mencionado, las páginas sucias son las que se han actualizado en la memoria pero no en su versión en disco.
4. Utiliza un esquema de revisión difusa que solo registra información sobre las páginas sucias e información asociada, y no requiere siquiera la escritura de las páginas sucias a disco. Saca las páginas sucias continuamente en segundo plano, en lugar de escribirlas durante los puntos de revisión.

En el resto de esta sección se presenta una visión general de ARIES. Las notas bibliográficas incluyen referencias que proporcionan una descripción completa de ARIES.

16.8.1. Estructuras de datos

Cada registro del registro histórico de ARIES tiene un **número de secuencia del registro histórico (NSR)** que lo identifica unívocamente. El número es conceptualmente tan solo un identificador lógico cuyo valor es mayor para los registros que aparecen después en el registro histórico. En la práctica, el NSR se genera de forma que también se puede usar para localizar el registro del registro histórico en disco. Normalmente, ARIES divide el registro histórico en varios archivos de registro histórico, cada uno con un número de archivo. Cuando un archivo crece hasta un determinado límite, ARIES añade los nuevos registros del registro histórico en un nuevo archivo; el nuevo archivo de registro histórico tiene un número de archivo que es 1 mayor que el anterior archivo. El NSR consiste en un número de archivo y un desplazamiento dentro del archivo.

Cada página también mantiene un identificador denominado **NSRPágina.** Cada vez que se aplica una operación (física o fisiológica) en la página, la operación almacena el NSR de su registro en el campo NSRPágina de la página. Durante la fase rehacer de la recuperación cualquier registro con un NSR menor o igual que el NSRPágina de la página no se debería ejecutar, ya que sus acciones ya están reflejadas en la página. En combinación con un esquema para el registro de los NSRPágina como parte de los puntos de revisión, que se presenta más adelante, ARIES puede evitar incluso leer muchas páginas cuyas operaciones registradas ya se han reflejado en el disco. Por tanto, el tiempo de recuperación se reduce significativamente.

El NSRPágina es esencial para asegurar la idempotencia en presencia de operaciones rehacer fisiológicas, ya que volver a aplicar una operación rehacer fisiológica que ya se haya aplicado a una página podría causar cambios incorrectos en esta.

Las páginas no se deberían enviar a disco mientras se esté realizando una actualización, dado que las operaciones fisiológicas no se pueden rehacer sobre el estado parcialmente actualizado de la página en disco. Por tanto, ARIES usa pestillos sobre las páginas de la memoria intermedia para evitar que se escriban en disco mientras se actualicen. Los pestillos de las páginas de la memoria intermedia solo se liberan cuando se completan las actualizaciones, y el registro del registro histórico para la actualización se haya escrito en el registro histórico.

Cada registro del registro histórico contiene también el NSR del registro anterior de la misma transacción. Este valor, almacenado en el campo NSRAnterior, permite que se encuentren los registros del registro histórico anteriores sin necesidad de leer el registro histórico completo. En ARIES hay registros especiales solo-rehacer generados durante el retroceso de transacciones, denominados **registros de compensación del registro histórico (RCR).** Cumplen el mismo objetivo que los registros del registro histórico de solo-rehacer en nuestro esquema de recuperación anterior. Además, los RCR cumplen el papel de los registro del registro histórico abortar-operación de nuestro esquema. Los RCR tienen un campo extra denominado DeshacerSiguienteNSR, que registra el NSR del registro que hay que deshacer a continuación cuando la transacción retrocede. Este campo sirve para el mismo propósito que el identificador de operaciones en el registro abortar-operación del esquema anterior, que ayuda a omitir los registros que ya hayan retrocedido.

La **TablaPáginasSucias** contiene una lista de páginas que se han actualizado en la memoria intermedia de la base de datos. Para cada página se almacena el NSRPágina y un campo denominado RegNSR que ayuda a identificar los registros que ya se han aplicado a la versión en disco de la página. Cuando se inserta una página en la TablaPáginasSucias (cuando se modifica por primera vez en el grupo de memorias intermedias) el valor de RegNSR se establece en el fin actual del registro histórico. Cada vez que se saca una página a disco, la página se elimina de la TablaPáginasSucias.

El **registro punto de revisión del registro histórico** contiene la TablaPáginasSucias y una lista de transacciones activas. Para cada transacción, el registro punto de revisión del registro histórico también anota ÚltimoNSR; el NSR del último registro escrito por la transacción. Una posición fija en el disco también anota el NSR del último registro punto de revisión del registro histórico (completado).

En la Figura 16.8 se muestran algunas estructuras de datos que se usan en ARIES. Los registros del registro histórico que se muestran en la figura tienen como prefijo su NSR; estos no se pueden almacenar explícitamente, pero en una implementación real se infieren de la posición en el registro histórico. El identificador de un elemento de datos en un registro se muestra en dos partes, por ejemplo 4894-1; la primera parte identifica la página y la segunda el registro dentro de la página (se supone una organización de registros de páginas con ranuras en la página). Fíjese que el registro histórico se muestra con los registros más recientes arriba, ya que los anteriores, que están en el disco, se muestran en la parte inferior de la figura.

Cada página (esté en la memoria intermedia o en el disco) tiene un campo NSRPágina asociado. Se puede verificar que el NSR del último registro que actualizó la página 4894 es el 7567. Comparando el NSRPágina de las páginas en la memoria intermedia con el NSRPágina para las correspondientes páginas en almacenamiento estable, se puede observar que la TablaPáginasSucias contiene entradas de todas las páginas en la memoria intermedia que han sido modificadas desde que se sacaron al almacenamiento estable. La entrada RegNSR de la TablaPáginasSucias refleja el NSR al final del registro histórico cuando la página se añade a la TablaPáginasSucias, y debería ser mayor o igual en almacenamiento al NSRPágina para esa página.

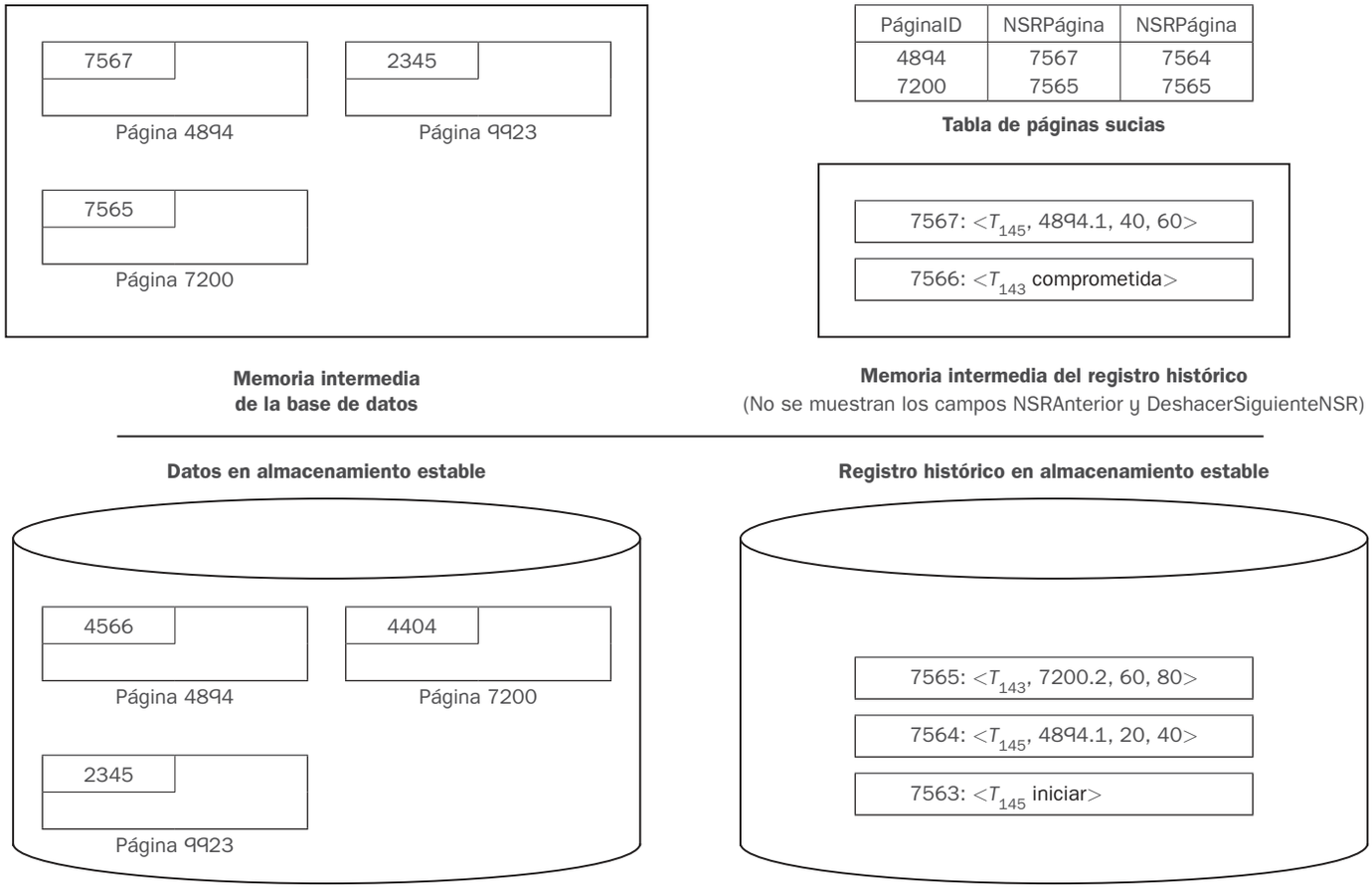


Figura 16.8. Estructuras de datos utilizadas en ARIES.

16.8.2. Algoritmo de recuperación

ARIES recupera de un fallo del sistema en tres fases:

- **Paso de análisis.** Este paso determina las transacciones que hay que deshacer, las páginas que están en estado sucias en el momento de la caída y el NSR en el que debería comenzar el paso rehacer.
- **Paso rehacer.** Este paso comienza en una posición determinada durante el análisis y realiza una operación rehacer, repitiendo la historia, para llevar a la base de datos al estado anterior al fallo.
- **Paso deshacer.** Este paso hace retroceder todas las transacciones incompletas en el momento del fallo.

16.8.2.1. Paso de análisis

El paso de análisis busca el último registro punto de revisión completado del registro histórico y lee la TablaPáginasSucias en este registro. A continuación establece RehacerNSR al mínimo RegistroNSR de las páginas de TablaPáginasSucias. Si no hay páginas sucias, establece RehacerNSR al NSR del registro punto de revisión del registro histórico. El paso rehacer comienza explorando el registro histórico desde RehacerNSR. Todos los registros anteriores a este punto ya se han aplicado a las páginas de la base de datos en el disco. El paso de análisis ajusta inicialmente la lista de transacciones que se deben deshacer, lista-deshacer, a la lista de transacciones en el registro punto de revisión del registro histórico. El paso de análisis también lee, a partir del registro punto de revisión del registro histórico, los NSR del último registro del registro histórico para cada transacción de la lista-deshacer.

El paso de análisis continúa examinando hacia delante desde el punto de revisión. Cada vez que encuentra un registro de una transacción que no esté en la lista-deshacer, añade la transacción a la lista-deshacer. Cada vez que encuentra un registro de fin de transacción, borra la transacción de la lista-deshacer.

Todas las transacciones que queden en la lista-deshacer al final del análisis se deben hacer retroceder posteriormente en el paso deshacer. El paso de análisis también almacena el último registro de cada transacción en la lista-deshacer, que se usa en el paso deshacer.

El paso de análisis también actualiza TablaPáginasSucias cada vez que encuentra un registro del registro histórico de la actualización de una página. Si la página no está en la TablaPáginasSucias, el paso de análisis la añade a ella y establece el RegistroNSR de la página al NSR del registro.

16.8.2.2. Paso rehacer

El paso rehacer repite la historia volviendo a ejecutar todas las acciones que no se hayan realizado en una página en disco. El paso rehacer examina el registro histórico hacia delante a partir de RehacerNSR. Cada vez que encuentra un registro de actualización realiza lo siguiente:

1. Si la página no está en la TablaPáginasSucias o el NSR del registro de actualización es menor que el RegistroNSR de la página de TablaPáginasSucias, entonces el paso rehacer omite el registro.
2. En caso contrario, el paso rehacer obtiene la página del disco y, si NSRPágina es menor que el NSR del registro, se rehace el registro.

Observe que si cualquiera de las comprobaciones es negativa, entonces los efectos del registro del registro histórico ya se han realizado en la página. Si la primera comprobación es negativa, ni siquiera es necesario extraer la página de disco para realizar la comprobación de NSRPágina.

16.8.2.3. Paso deshacer y retroceso de transacciones

El paso deshacer es relativamente simple. Realiza una exploración hacia atrás del registro histórico, deshaciendo todas las transacciones de la lista-deshacer. El paso deshacer examina solo los registros históricos de transacciones en la lista-deshacer; el último NSR registrado durante el paso de análisis sirve para encontrar el último registro para cada transacción en la lista-deshacer.

Cuando se encuentra un registro de actualización en el registro histórico, se usa para realizar un deshacer (bien para el retroceso de una transacción durante el procesamiento normal o durante el paso deshacer del reinicio). El paso deshacer genera un RCR que contiene la acción deshacer realizada (que debe ser fisiológica). Establece el DeshacerSiguienteNSR del RCR al valor NSRAnterior del registro de actualización.

Si se encuentra un RCR, el valor de DeshacerSiguienteNSR indica el NSR del siguiente registro a deshacer para esa transacción; los registros posteriores para dicha transacción ya se han retrocedido. Para registros distintos de RCR, el campo NSRAnterior del registro indica el NSR del siguiente registro que hay que deshacer para esa transacción. El siguiente registro a procesar en cada paso del paso deshacer es el máximo, de entre todas las transacciones de la lisa-deshacer, de los registros NSR.

En la Figura 16.9 se muestran las acciones de recuperación que se realizan en ARIES, con un ejemplo de registro. Suponga que el último punto de revisión en el disco apunta al registro de revisión con el NSR 7568. Los valores NSR de los registros del registro histórico se muestran en la figura con flechas, mientras que los valores de DeshacerSiguienteNSR se muestran usando una flecha discontinua para el registro de compensación, con el NSR 7565, en la figura. El paso de análisis empezaría desde el NSR 7568, y cuando termina NSRDeshacer será 7564. Por tanto, el paso deshacer debe empezar en el registro con el NSR 7564. Fíjese que este NSR es menor que el NSR del registro de revisión, ya que el algoritmo de punto de revisión de ARIES no saca las páginas modificadas hacia la memoria estable. La TablaPáginasSucias al final del análisis incluirá las páginas 4894, 7200 del registro de punto de revisión y 2390, que se actualiza por el registro con el NSR 7570. Al final del paso de análisis, la lista de transacciones que hay que deshacer consta solo de la T_{145}, en este ejemplo.

El paso rehacer para el ejemplo anterior empieza desde el NSR 7564 y realiza las operaciones rehacer de los registros del registro histórico cuyas páginas aparezcan en la TablaPáginasSucias. El paso deshacer necesita deshacer solo la transacción T_{145} y, por tanto, empieza desde el NSRAnterior con valor 7567 y continúa hacia atrás hasta que se encuentra el registro <T_{145} iniciar> en el NSR 7563.

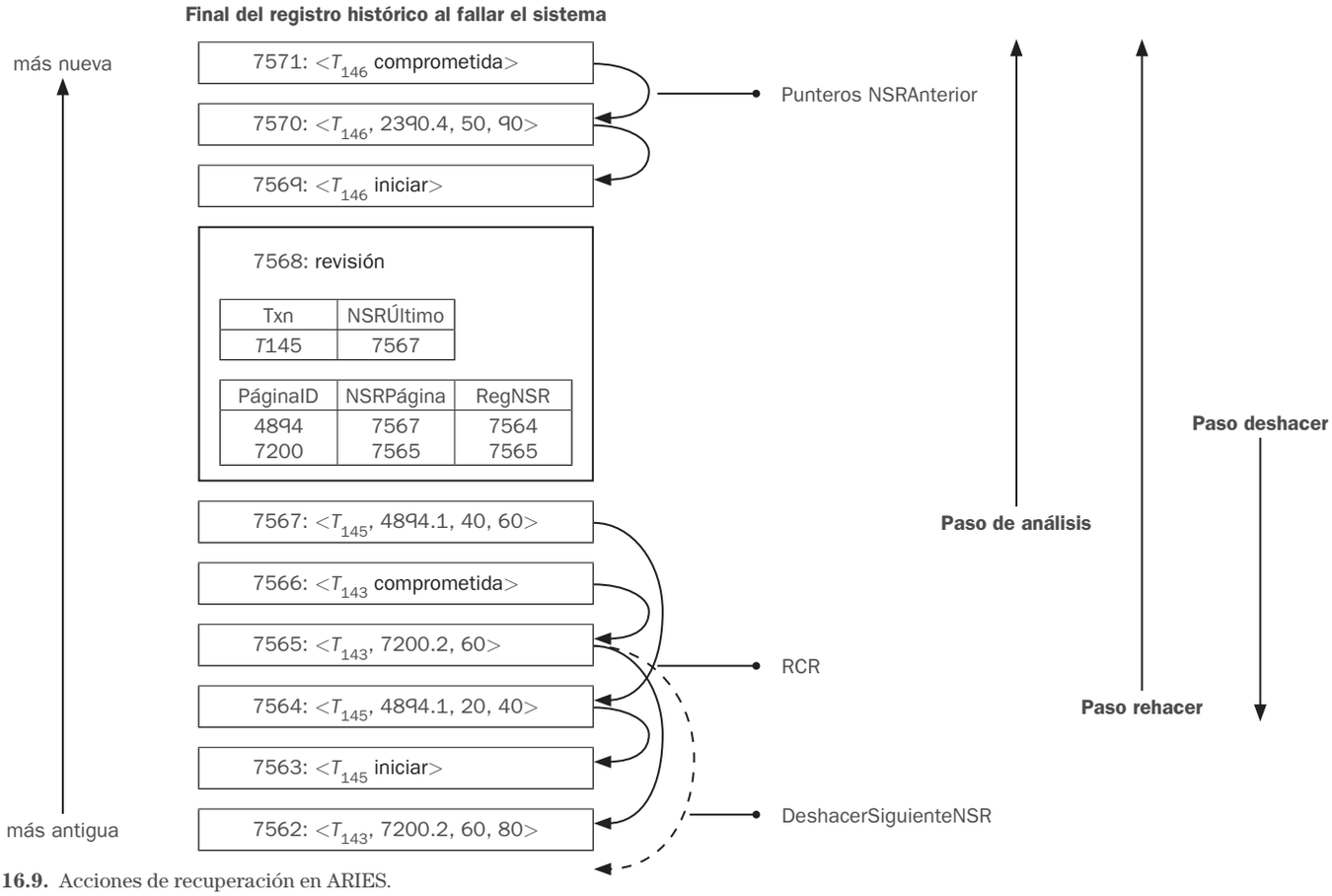


Figura 16.9. Acciones de recuperación en ARIES.

16.8.3. Otras características

Algunas de las características que proporciona ARIES son:

- **Acciones superiores anidadas.** ARIES permite el registro histórico de las operaciones que no se deberían deshacer incluso si la transacción se retrocede; por ejemplo, si una transacción asigna una página a una relación, aunque la transacción se retroceda la asignación de la página no se debería deshacer ya que otras transacciones podrían haber guardado registros en dicha página. Estas operaciones que no se deberían deshacer se denominan acciones superiores anidadas. Estas operaciones se pueden modelar como operaciones cuyas acciones de deshacer no hacen nada. En ARIES, tales operaciones se implementan creando NSR vacío cuyo DeshacerSiguienteNSR se establece de forma que la operación de retroceso de transacciones se salta el registro generado por la operación.
- **Independencia de recuperación.** Algunas páginas se pueden recuperar independientemente de otras, de forma que se pueden usar incluso cuando se estén recuperando otras. Si fallan algunas páginas del disco se pueden recuperar sin parar el procesamiento de transacciones en otras páginas.
- **Puntos de almacenamiento.** Las transacciones pueden registrar puntos de almacenamiento y pueden retroceder parcialmente hasta un punto de almacenamiento. Esto puede ser muy útil en el manejo de interbloqueos, dado que las transacciones pueden retroceder hasta un punto que permita la liberación de los bloqueos requeridos y luego reiniciarse desde ese punto.

  Los programadores pueden usar los puntos de almacenamiento para deshacer parcialmente una transacción y después continuar la ejecución; este enfoque puede ser útil para manejar ciertos tipos de errores que se detecten durante la ejecución de la transacción.
- **Bloqueo de grano fino.** El algoritmo de recuperación ARIES se puede usar con algoritmos de control de concurrencia de índices que permiten el bloqueo en el nivel de tuplas de los índices, en lugar del bloqueo en el nivel de las páginas, lo que aumenta significativamente la concurrencia.
- **Optimizaciones de la recuperación.** La TablaPáginasSucias se puede usar para obtener por adelantado páginas durante la operación rehacer, en lugar de extraer una página solo cuando el sistema encuentra un registro del registro histórico a aplicar a la página. La operación rehacer-no-válidos también es posible. Esta operación se puede posponer sobre una página que se vaya a extraer del disco y realizarse cuando se extraiga. Mientras tanto se pueden seguir procesando otros registros.

En resumen, el algoritmo ARIES es un algoritmo de recuperación actual que incorpora varias optimizaciones diseñadas para mejorar la concurrencia, reducir la sobrecarga por el registro histórico y reducir el tiempo de recuperación.

16.9. Sistemas remotos de copias de seguridad

Los sistemas tradicionales de procesamiento de transacciones son sistemas centralizados o sistemas cliente-servidor. Esos sistemas son vulnerables frente a desastres ambientales como el fuego, las inundaciones o los terremotos. Hay una necesidad creciente de sistemas de procesamiento de transacciones que ofrezcan una disponibilidad elevada y que puedan funcionar pese a los desastres ambientales. Estos sistemas deben proporcionar una **alta disponibilidad;** es decir, el tiempo en que el sistema no es utilizable debe ser extremadamente reducido.

Se puede obtener una disponibilidad elevada realizando el procesamiento de transacciones en un sitio, denominado **sitio principal,** pero tener un sitio de **copia de seguridad remota** en el que se repliquen todos los datos del sitio principal. El sitio de copia de seguridad remota a veces se denomina también **sitio secundario.** El sitio remoto debe mantenerse sincronizado con el sitio principal, ya que las actualizaciones se realizan en el sitio principal. La sincronización se obtiene enviando todos los registros del registro histórico desde el sitio principal al sitio remoto de copia de seguridad. El sitio remoto copia de seguridad debe hallarse físicamente separado del principal —por ejemplo, se puede ubicar en otra provincia — para que una catástrofe en el sitio principal no afecte al sitio remoto copia de seguridad. En la Figura 16.10 se muestra la arquitectura de los sistemas para copias de seguridad remotas.

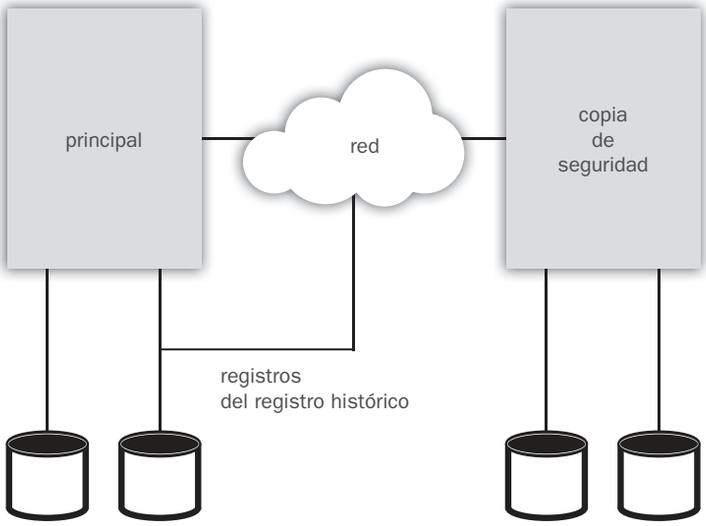


Figura 16.10. Arquitectura de los sistemas remotos de copias de seguridad.

Cuando falla el sitio principal, el sitio remoto asume el procesamiento. En primer lugar, sin embargo, lleva a cabo la recuperación utilizando su copia (tal vez anticuada) de los datos del sitio principal y de los registros del registro histórico recibidos del mismo. En realidad, el sitio remoto lleva a cabo acciones de recuperación que se hubieran llevado a cabo en el sitio principal cuando este se hubiera recuperado. Se pueden utilizar los algoritmos estándar de recuperación para la recuperación en el sitio remoto con pocas modificaciones. Una vez realizada la recuperación, el sitio remoto comienza a procesar transacciones.

La disponibilidad aumenta mucho en comparación a los sistemas con un solo sitio, dado que el sistema puede recuperarse aunque se pierdan todos los datos del sitio principal. El rendimiento de los sistemas de copias de seguridad remota es mejor que el de los sistemas distribuidos con compromiso de dos fases.

Al diseñar sistemas de copias de seguridad remotas se deben abordar varios aspectos, tales como:

- **Detección de fallos.** Es importante que el sistema de copias de seguridad remotas detecte que el sitio principal ha fallado. El fallo de las líneas de comunicación puede hacer creer al sitio remoto de copia de seguridad que el sitio principal ha fallado. Para evitar este problema hay que mantener varios enlaces de comunicaciones con modos de fallo independientes entre el sitio principal y el sitio remoto de copia de seguridad. Por ejemplo, además de la conexión de red puede haber otra conexión mediante módem por línea telefónica con servicio suministrado por diferentes compañías de telecomunicaciones. Estas conexiones pueden complementarse con la intervención manual de operadores, que se pueden comunicar por vía telefónica.

- **Transferencia del control.** Cuando el sitio principal falla, el sitio de copia de seguridad asume el procesamiento y se transforma en el nuevo sitio principal. Cuando el sitio principal original se recupera puede desempeñar el papel de sitio de copias de seguridad remotas o volver a asumir el papel de sitio principal. En cualquiera de los casos, el sitio principal anterior debe recibir un registro histórico de actualizaciones realizado por el sitio de copia de seguridad mientras el sitio principal antiguo estaba fuera de servicio.

  La manera más sencilla de transferir el control es que el sitio principal anterior reciba el registro histórico de operaciones rehacer del sitio de copia de seguridad antiguo y se ponga al día con las actualizaciones aplicándolas de manera local. El sitio principal antiguo puede entonces actuar como sitio remoto de copia de seguridad. Si hay que devolver el control, el sitio remoto anterior puede simular que ha fallado, lo que da lugar a que el sitio principal anterior asuma el control.
- **Tiempo de recuperación.** Si el registro histórico del sitio remoto de copia de seguridad se hace grande, la recuperación puede tardar mucho. El sitio remoto de copia de seguridad puede procesar de manera periódica los registros rehacer del registro histórico que haya recibido y realizar un punto de revisión, de manera que se puedan borrar las partes más antiguas del registro histórico. Como consecuencia es posible reducir el retraso antes de que el sitio remoto de copia de seguridad asuma el control.

  Una configuración de **relevo en caliente** puede hacer la toma del control por el sitio de copia de seguridad casi instantáneo. En esta configuración el sitio remoto copia de seguridad procesa los registros rehacer del registro histórico según llegan, y aplica las actualizaciones de manera local. Tan pronto como se detecta el fallo del sitio principal, el sitio de copia de seguridad completa la recuperación haciendo retroceder las transacciones incompletas y queda preparado para procesar las nuevas.
- **Tiempo de compromiso.** Para asegurar que las actualizaciones de una transacción comprometida sean duraderas no se debe declarar comprometida una transacción hasta que sus registros del registro histórico hayan alcanzado el sitio de copia de seguridad. Este retraso puede dar lugar a una espera más prolongada para comprometer la transacción y, en consecuencia, algunos sistemas permiten grados inferiores de durabilidad. Los grados de durabilidad pueden clasificarse de la siguiente manera:
  - **Uno seguro.** Las transacciones se comprometen tan pronto como sus registros de compromiso del registro histórico se escriben en un almacenamiento estable en el sitio principal.

    El problema de este esquema es que puede que las actualizaciones de una transacción comprometida no hayan alcanzado el sitio de copia de seguridad cuando este asuma el control del procesamiento. Por tanto, puede parecer que las actualizaciones se han perdido. Cuando se recupera el sitio principal, las actualizaciones perdidas no se pueden mezclar directamente, dado que pueden entrar en conflicto con actualizaciones posteriores llevadas a cabo en el sitio de copia de seguridad. Por tanto, puede que se necesite la intervención humana para devolver a la base de datos a un estado consistente.
  - **Dos muy seguro.** Las transacciones se comprometen tan pronto como sus registros de compromiso del registro histórico se escriben en un almacenamiento estable en el sitio principal y en el sitio de copia de seguridad.

    El problema de este esquema es que el procesamiento de transacciones no puede continuar si alguno de los puntos no está operativo. Por tanto, la disponibilidad es realmente menor que en el caso de un solo sitio, aunque la posibilidad de la pérdida de datos es mucho más reducida.
  - **Dos seguro.** Este esquema es idéntico al esquema dos muy seguro si tanto el sitio principal como el sitio de copia de seguridad están activos. Si solo está activo el sitio principal se permite que la transacción se comprometa tan pronto como su registro de compromiso del registro histórico se escriba en un almacenamiento estable en el sitio principal.

    Este esquema proporciona mejor disponibilidad que el esquema dos muy seguro, al tiempo que evita el problema de las transacciones perdidas afrontado por el esquema uno seguro. Da lugar a un compromiso más lento que el esquema uno seguro pero las ventajas generalmente superan los inconvenientes.

Varios sistemas comerciales de disco compartido proporcionan un nivel de tolerancia de fallos intermedio entre el de los sistemas centralizados y el de los sistemas remotos para copias de seguridad. En estos sistemas el fallo de una CPU no da lugar al fallo del sistema. En su lugar, otra CPU asume el control y lleva a cabo la recuperación. Las acciones de recuperación incluyen el retroceso de las transacciones que se ejecutaban en la CPU que falló y la recuperación de los bloqueos mantenidos por dichas transacciones. Dado que los datos se hallan en un disco compartido, no hace falta transferir registros del registro histórico. Sin embargo, se deberían proteger los datos contra el fallo del disco utilizando, por ejemplo, una organización de discos RAID.

Una forma alternativa de conseguir alta disponibilidad es usar una base de datos distribuida con los datos replicados en más de un sitio. Son necesarias transacciones para actualizar todas las réplicas de cualquier elemento de datos que actualicen. Las bases de datos distribuidas, incluyendo la réplica, se estudian en el Capítulo 19.

16.10. Resumen

- Los sistemas informáticos, al igual que cualquier otro dispositivo eléctrico o mecánico, están sujetos a fallos. Estos fallos se producen por diferentes motivos incluyendo fallos de disco, cortes de corriente o fallos en el software.

  En cada uno de estos casos puede perderse información concerniente a la base de datos.
- Las transacciones pueden fallar, además de por un fallo del sistema, por otras razones como una violación de las restricciones de integridad o interbloqueos.
- El esquema de recuperación es una parte integrante del sistema de base de datos, el cual es responsable de la detección de fallos y del restablecimiento de un estado de la base de datos anterior al momento de producirse el fallo.
- Los diferentes tipos de almacenamiento en una computadora son el volátil, el no volátil y el almacenamiento estable. Los datos del almacenamiento volátil, como ocurre con los guardados en la memoria RAM, se pierden cuando la computadora falla. Los datos del almacenamiento no volátil, como los guardados en un disco, no se pierden cuando la computadora falla, pero pueden perderse ocasionalmente debido a fallos en el disco. Los datos del almacenamiento estable nunca se pierden.
- El almacenamiento estable de acceso en tiempo real puede aproximarse con discos con imagen u otras formas de RAID que proporcionan almacenamiento redundante de datos. El almacenamiento estable, sin conexión o de archivo, puede consistir en una serie de copias en cinta de los datos guardadas en un lugar físicamente seguro.

- El estado del sistema de base de datos puede no volver a ser consistente en caso de producirse un fallo; es decir, puede no reflejar el estado del mundo que debe capturar la base de datos. Para mantener la consistencia es necesario que cada transacción sea atómica. Garantizar las propiedades de atomicidad y durabilidad es responsabilidad del esquema de recuperación.
- En los esquemas basados en registro histórico todas las modificaciones se escriben en el registro histórico, que debe estar guardado en almacenamiento estable. Se considera que una transacción está comprometida cuando su último registro del registro histórico, que es el registro comprometido de la transacción, se ha escrito en almacenamiento estable.
- Los registros del registro histórico contienen los valores anterior y nuevo de todos los elementos de datos actualizados. Los valores nuevos se usan en el caso de que se necesite rehacer las transacciones tras un fallo del sistema. Los valores anteriores se usan para retroceder las actualizaciones de la transacción si esta aborta durante la operación normal, así como para retroceder las actualizaciones de las transacciones en caso de que el sistema falle antes de que la transacción pase a comprometida.
- En el esquema de modificación diferida, durante la ejecución de una transacción, se difieren todas las operaciones escribir hasta que la transacción se compromete parcialmente, momento en el que se utiliza la información del registro histórico asociada con la transacción para ejecutar las escrituras diferidas. Con las modificaciones diferidas, los registros del registro histórico no necesitan contener valores anteriores de los elementos de datos actualizados.
- Puede usarse la técnica de los puntos de revisión para reducir la sobrecarga que conlleva la búsqueda en el registro histórico y rehacer las transacciones.
- Los algoritmos modernos de recuperación se basan en el concepto de repetición de la historia, en el que todas las acciones que se realizan durante la operación normal (desde que se ha completado el último punto de revisión) se vuelven a hacer durante el paso de rehacer, para la recuperación. La repetición de la historia restaura el estado del sistema al que era en el momento en que salió a almacenamiento estable el último registro del registro histórico antes de que el sistema fallase. La operación deshacer se realiza desde este estado, ejecutando un paso de deshacer que procesa los registros del registro histórico de las transacciones incompletas en orden inverso.
- Deshacer una transacción incompleta escribe registros de solo-rehacer especiales en el registro histórico, y un registro abortar. Después, la transacción se puede considerar terminada y no se volverá a deshacer de nuevo.
- El procesamiento de transacciones se basa en un modelo de almacenamiento en el que la memoria principal contiene una memoria intermedia para el registro histórico, una memoria intermedia para la base de datos y una memoria intermedia para el sistema. La memoria intermedia del sistema alberga páginas de código objeto del sistema y áreas de trabajo local de las transacciones.
- Una implementación eficiente de un esquema de recuperación de datos requiere que el número de escrituras en la base de datos y en almacenamiento estable sea mínimo. Los registros del registro histórico pueden guardarse inicialmente en la memoria intermedia del registro histórico en almacenamiento volátil, pero se deben copiar en almacenamiento estable cuando se da una de estas dos condiciones:
  - Deben escribirse en almacenamiento estable todos los registros del registro histórico pertenecientes a la transacción T_i antes de que el registro $<T_i$ comprometida$>$ se pueda escribir en almacenamiento estable.
  - Deben escribirse en almacenamiento estable todos los registros del registro histórico pertenecientes a los datos de un bloque antes de que ese bloque de datos se escriba desde la memoria principal a la base de datos (en almacenamiento no volátil).
- Las técnicas avanzadas de recuperación utilizan técnicas de bloqueo de alta concurrencia, como las utilizadas para el control de concurrencia con árboles B^+. Estas técnicas permiten la liberación rápida de bloqueo de bajo nivel que adquieren las operaciones como insertar o borrar, lo que permite que otras transacciones realicen otras operaciones. Cuando se liberan los bloqueos de bajo nivel ya no se puede hacer una operación deshacer, en lugar de un deshacer lógico, se necesita un borrado para deshacer una inserción. Las transacciones mantienen los bloqueos de alto nivel que aseguran que las transacciones concurrentes no realizan acciones que hagan imposible la operación deshacer lógica.
- Para recuperarse de los fallos que dan lugar a la pérdida de almacenamiento no volátil, debe realizarse un volcado periódicamente (por ejemplo, una vez al día) del contenido entero de la base de datos en almacenamiento estable. Se usará el último volcado para devolver a la base de datos a un estado consistente previo cuando se produzca un fallo que conduzca a la pérdida de algún bloque físico de la base de datos. Una vez realizada esta operación se utilizará el registro histórico para llevar a la base de datos al estado consistente más reciente.
- El esquema de recuperación ARIES es un esquema actual que soporta varias características para proporcionar mayor concurrencia, reducir la sobrecarga del registro histórico y permitir operaciones deshacer lógicas. También se basa en la repetición de la historia y permite las operaciones deshacer lógicas. El esquema da salida a las páginas continuamente y no necesita procesar todas las páginas en el momento de un punto de revisión. Usa números de secuencia del registro histórico (NSR) para implementar varias optimizaciones que reducen el tiempo de recuperación.
- Los sistemas de copias de seguridad remotas proporcionan un alto nivel de disponibilidad, permitiendo que continúe el procesamiento de transacciones incluso si se destruye el sitio primario por fuego, inundación o terremoto. Si el sitio principal falla, el sitio remoto puede encargarse del procesamiento de las transacciones, tras ejecutar ciertas acciones de recuperación.

Términos de repaso

- Esquema de recuperación.
- Clasificación de los fallos.
  - Fallo de transacción.
  - Error lógico.
  - Error del sistema.
  - Fallo del sistema.
  - Fallo de transferencia de datos.
- Supuesto de fallo-parada.
- Fallo de disco.
- Tipos de almacenamiento.
  - Volátil.
  - No volátil.
  - Estable.
- Memoria intermedia de disco.

- Bloques.
  - Físicos.
  - De memoria intermedia.
- Escritura forzada.
- Recuperación basada en el registro histórico.
- Registro histórico.
- Registros del registro histórico.
- Registro actualizar del registro histórico.
- Modificación diferida.
- Modificación inmediata.
- Modificaciones no comprometidas.
- Puntos de revisión.
- Algoritmo de recuperación.
- Recuperación al reiniciar.
- Retroceso de transacciones.
- Deshacer físico.
- Registro histórico físico.
- Puntos de revisión.
- Fase de rehacer.
- Fase de deshacer.
- Repetición de la historia.
- Gestión de la memoria intermedia.
- Registro histórico con memoria intermedia.
- Registro de escritura anticipada (REA).
- Forzar el registro histórico.
- Memoria intermedia de la base de datos.
- Pestillos.
- Sistema operativo y gestión de la memoria intermedia.
- Puntos de revisión difusos.
- Liberación rápida de bloqueo.
- Operaciones lógicas.
- Registro histórico lógico.
- Deshacer lógico.
- Pérdida de almacenamiento no volátil.
- Volcado de archivo.
- Volcado difuso.
- ARIES.
  - Número de secuencia del registro histórico (NSR).
  - NSRPágina.
  - Operación rehacer fisiológica.
  - Registros de compensación del registro histórico (RCR).
  - TablaPáginasSucias.
  - Registro punto de revisión.
  - Fase de análisis.
  - Fase de rehacer.
  - Fase de deshacer.
- Alta disponibilidad.
- Sistemas de copia de seguridad remota.
  - Sitio principal.
  - Sitio remoto de copia de seguridad.
  - Sitio secundario.
- Detección de fallos.
- Transferencia del control.
- Tiempo de recuperación.
- Configuración de relevo en caliente.
- Tiempo de compromiso.
  - Uno seguro.
  - Dos muy seguro.
  - Dos seguro.

Ejercicios prácticos

16.1. Explique por qué los registros del registro histórico para las transacciones de la lista-deshacer deben procesarse en orden inverso, mientras las de rehacer se realizan en orden directo.

16.2. Explique el objetivo del mecanismo de puntos de revisión. ¿Cada cuánto se deben realizar? Indique cómo afecta la frecuencia de los puntos de revisión a:
- Rendimiento del sistema cuando no ocurren fallos.
- Tiempo que se tarda en recuperar desde un fallo del sistema.
- Tiempo que se tarda en recuperar desde un fallo en un medio (disco).

16.3. Algunos sistemas de bases de datos permiten que el administrador elija entre dos formas del registro histórico: el *normal*, que se utiliza para recuperar el sistema de un fallo, y el de *archivo*, que se utiliza para recuperar un fallo en un medio (disco). Indique cuándo se puede borrar un registro en cada uno de los casos utilizando el algoritmo de la Sección 16.4.

16.4. Describa cómo modificar el algoritmo de recuperación de la Sección 16.4 para implementar puntos de recuperación y realizar retrocesos a un punto de almacenamiento (se describen en la Sección 16.8.3).

16.5. Suponga que se usa la técnica de modificación diferida en la base de datos.

a. ¿Se sigue necesitando la parte de valor anterior de un registro de actualización del registro histórico? Indique por qué sí o por qué no.

b. Si los valores anteriores no se guardan en los registros de actualización del registro histórico, claramente no se puede realizar la operación deshacer de una transacción. ¿Cómo tendría entonces que modificarse la fase de rehacer de la recuperación?

c. La modificación diferida se puede implementar manteniendo los elementos de datos actualizados en la memoria local de las transacciones y leyendo los elementos de datos que no se hayan actualizado directamente de la memoria intermedia de la base de datos. Sugiera una implementación eficiente de la lectura de los elementos de datos que asegure que la transacción puede ver sus propias actualizaciones.

d. ¿Qué problema puede llegar a ocurrir con la técnica anterior, si las transacciones realizan actualizaciones de mucho tiempo?

16.6. El esquema de paginación en la sombra requiere que se copie la tabla de página. Suponga que la tabla de página se representa como un árbol B^+.

a. Sugiera cómo compartir tantos nodos como se pueda entre la nueva copia y la copia en la sombra del árbol B^+, suponiendo que las actualizaciones se hacen solo en las entradas hoja, sin inserciones ni borrados.

b. Incluso con la optimización anterior, el registro histórico es mucho menos costoso que un esquema de copia en la sombra para las transacciones que realicen actualizaciones pequeñas. Explique por qué.

16.7. Suponga que, de forma incorrecta, se modifica el algoritmo de recuperación de la Sección 16.4 para no registrar las acciones que se realizan durante el retroceso de las transacciones. Cuando se recupera de un fallo del sistema, las transacciones que se retrocedieron anteriormente se deberían incluir en la lista-deshacer y volverse a retroceder. Indique un ejemplo que muestre las acciones que se llevan a cabo durante la fase de deshacer de la recuperación que podrían generar un estado incorrecto de la base de datos. (Sugerencia: suponga un elemento de datos actualizado por una transacción abortada y después actualizado por una transacción que pasa a comprometida).

16.8. El espacio de disco que se asigna a un archivo como resultado de una transacción no se debería liberar incluso aunque la transacción se retroceda. Explique por qué, y explique cómo asegura ARIES que estas acciones no se retroceden.

16.9. Suponga que una transacción borra un registro y el espacio libre generado se asigna a un registro que inserta otra transacción, incluso antes de que la primera transacción pase a comprometida.

a. ¿Qué problema puede producirse si la primera transacción tiene que retrocederse?

b. ¿Seguiría suponiendo un problema si se utilizase bloqueo en el nivel de página en lugar de en el nivel de tupla?

c. Sugiera cómo resolver este problema mientras se dispone de bloqueo en el nivel de tupla, realizando el registro de acciones post-comprometida en registros especiales del registro histórico y ejecutándolos tras el paso a comprometida. Asegúrese de que el esquema garantiza que estas acciones se realizan exactamente una vez.

16.10. Explique las razones por las que la recuperación en transacciones interactivas es más difícil de tratar que la recuperación en transacciones por lotes. ¿Existe alguna forma sencilla de tratar esta dificultad?

(Sugerencia: considere una transacción de un cajero automático por la que se retira dinero).

16.11. A veces hay que deshacer una transacción después de que se haya comprometido porque se ejecutó erróneamente, debido por ejemplo a la introducción incorrecta de datos en un cajero.

a. Indique un ejemplo que demuestre que el uso de un mecanismo normal para deshacer esta transacción podría conducir a un estado inconsistente.

b. Una forma de manejar esta situación es llevar la base de datos a un estado anterior al compromiso de la transacción errónea (denominado recuperación *a un instante*). En este esquema se deshacen los efectos de las transacciones comprometidas después.

Sugiera una modificación del mecanismo de recuperación avanzada para implementar la recuperación a un instante.

c. Las transacciones correctas posteriores se pueden volver a ejecutar lógicamente, pero no se pueden reejecutar usando sus registros del registro histórico. ¿Por qué?

Ejercicios

16.12. Explique la diferencia en cuanto al coste de E/S entre los tres tipos de almacenamiento: volátil, no volátil y estable.

16.13. El almacenamiento estable no se puede implementar.

a. Explique el motivo.

b. Explique cómo tratan este problema los sistemas de las bases de datos.

16.14. Explique cómo el gestor de la memoria intermedia puede conducir a la base de datos a un estado inconsistente si algunos registros del registro histórico pertenecientes a un bloque no se escriben en almacenamiento estable antes de escribir en el disco el citado bloque.

16.15. Describa las desventajas de las políticas de gestión no-apropiación y memoria intermedia forzada.

16.16. El registro de rehacer fisiológico puede reducir la sobrecarga del registro histórico de forma significativa, especialmente con una organización de registros de página con ranuras.

Explique por qué.

16.17. Explique por qué se usa ampliamente el registro de deshacer lógico, mientras que el registro de rehacer lógico (distinto del registro de rehacer fisiológico) rara vez se utiliza.

16.18. Considere el registro histórico de la Figura 16.5. Suponga que se produce un fallo del sistema justo antes de que se escriba el registro $<T_0$ abortada$>$.

Explique qué debería ocurrir durante la recuperación.

16.19. Suponga que existe una transacción que se ha estado ejecutando durante un tiempo extremadamente largo, pero ha realizado unas pocas actualizaciones.

a. ¿Qué efecto tendrá la transacción en el tiempo de recuperación con el algoritmo de la Sección 16.4 y con el algoritmo de recuperación de ARIES?

b. ¿Qué efecto tendrá la transacción en el borrado de registros antiguos del registro histórico?

16.20. Considere el registro histórico de la Figura 16.6. Suponga que se produce un fallo del sistema durante la recuperación, justo antes de que se escriba el registro de la operación abortar en el registro histórico para la operación O_1. Explique qué debería ocurrir cuando comience de nuevo la recuperación del sistema.

16.21. Compare la recuperación basada en registro con el esquema de copia en la sombra en términos de sus sobrecargas, para aquellos casos en que se añaden datos a páginas de disco recién asignadas (es decir, no existen valores anteriores que haya que restaurar en el caso de que la transacción aborte).

16.22. En el algoritmo de recuperación de ARIES:

a. Si al comienzo del paso de análisis una página no se encuentra en la tabla de páginas sucias del punto de revisión, ¿será necesario aplicar los registros de rehacer al mismo? Explique por qué.

b. ¿Qué es el RegNST y cómo se usa para minimizar los rehacer innecesarios?

16.23. Explique la diferencia entre un fallo del sistema y un «desastre».

16.24. Para cada uno de los siguientes requisitos identifique la mejor opción del grado de durabilidad en un sistema de copias de seguridad remotas.

a. Se debe evitar la pérdida de datos pero se puede tolerar alguna pérdida de disponibilidad.

b. El compromiso de transacciones se debe realizar rápidamente, incluso perdiendo algunas transacciones comprometidas en caso de desastre.

c. Se requiere un alto grado de disponibilidad y durabilidad, pero es aceptable un mayor tiempo de ejecución para el protocolo de compromiso de transacciones.

16.25. El sistema de bases de datos de Oracle usa registros de deshacer para proporcionar una vista instantánea de la base de datos bajo aislamiento de instantáneas. La vista instantánea que ve la transacción T_i refleja las actualizaciones de todas las transacciones comprometidas cuando T_i comenzó y las actualizaciones de T_i; las actualizaciones del resto de transacciones no son visibles para T_i.

Describa un esquema para el gestor de memoria intermedia en el que se proporcione una vista instantánea de las páginas de la memoria intermedia. Incluya los detalles de cómo usar el registro histórico para generar la vista instantánea. Puede suponer que las operaciones, así como las acciones de deshacer, solo afecten a una página.

Notas bibliográficas

El libro de Gray y Reuter [1993] constituye una excelente fuente de información sobre la recuperación, incluyendo interesantes implementaciones y detalles históricos. Bernstein y Goodman [1981] es uno de los primeros libros de texto con información sobre control de concurrencia y recuperación.

En Gray et ál. [1981] se presenta una visión de conjunto del esquema de recuperación de System R. En Gray [1978], Lindsay et ál. [1980] y Verhofstad [1978] pueden encontrarse guías de aprendizaje y visiones de conjunto sobre varias técnicas de recuperación para sistemas de bases de datos. Los conceptos de punto de revisión difusa y volcado difuso se describen en Lindsay et ál. [1980]. Haerder y Reuter [1983] ofrecen una comprensible presentación de los principios de la recuperación.

La situación actual de los métodos de recuperación se ilustra mejor con el método de recuperación ARIES, descrito en Mohan et ál. [1992] y en Mohan [1990b]. Mohan y Levine [1992] presentan ARIES IM, una extensión de ARIES para optimizar el control de concurrencia y recuperación de árboles B^+ usando registros de deshacer lógicos del registro histórico. ARIES y sus variantes se usan en varios productos de bases de datos, incluyendo DB2 de IBM y SQL Server de Microsoft. La recuperación en Oracle se describe en Lahiri et ál. [2001].

Mohan y Levine [1992] y Mohan [1993] proporcionan técnicas de recuperación especializadas para estructuras con índices; Mohan y Narang [1994] describen técnicas de recuperación para arquitecturas cliente-servidor, mientras que Mohan y Narang [1992] describen técnicas de recuperación para arquitecturas de bases de datos paralelas.

En Weikum [1991] se describe una versión generalizada de la teoría de la secuencialidad, con bloqueos de bajo nivel de corta duración durante las operaciones, combinados con bloqueos de alto nivel de duración más larga. En la Sección 16.7.3 se abordó el requisito de que una operación debería adquirir todos los bloqueos de bajo nivel que fueron necesarios para el deshacer lógico de la operación. Este requisito se puede relajar realizando primero todas las operaciones de deshacer físicas, antes de realizar ninguna de las operaciones de deshacer lógicas. Una versión generalizada de esta idea, llamada recuperación multinivel, se presenta en Weikum et ál. [1990], y permite varios niveles de operaciones lógicas, con pasos de deshacer nivel por nivel durante la recuperación.

King et ál. [1991] y Polyzois y García-Molina [1994] presentan las copias de seguridad remotas para recuperación de desastres.

Parte 5
Arquitectura de sistemas

La arquitectura de los sistemas de bases de datos está enormemente influida por el sistema informático subyacente en el que se ejecuta el sistema de bases de datos. Los sistemas de bases de datos pueden ser centralizados, donde una máquina que hace de servidor ejecuta operaciones en la base de datos. Los sistemas de bases de datos también pueden diseñarse para explotar las arquitecturas paralelas de computadoras. Las bases de datos distribuidas abarcan muchas máquinas separadas geográficamente.

En el Capítulo 17 se empieza tratando las arquitecturas de los sistemas de bases de datos que se ejecutan en sistemas servidores, que se utilizan en arquitecturas centralizadas y cliente-servidor. En este capítulo se tratan los diferentes procesos que juntos implementan la funcionalidad de la base de datos. Después, se estudian las arquitecturas paralelas de computadoras y las arquitecturas paralelas de bases de datos diseñadas para diferentes tipos de computadoras paralelas. Finalmente, el capítulo trata asuntos arquitectónicos para la construcción de un sistema distribuido de bases de datos.

En el Capítulo 18 se describe la forma en que varias acciones de una base de datos, en particular el procesamiento de consultas, se pueden implementar para explotar el procesamiento paralelo.

En el Capítulo 19 se presentan varias cuestiones que surgen en una base de datos distribuida, y se describe cómo tratar cada cuestión. Estas cuestiones incluyen cómo almacenar datos, cómo asegurar la atomicidad de las transacciones que se ejecutan en varios emplazamientos, cómo realizar el control de concurrencia y cómo proporcionar alta disponibilidad ante la presencia de fallos. En este capítulo también se estudian los sistemas de almacenamiento basados en la nube, el procesamiento distribuido de consultas y los sistemas de directorio.

Arquitecturas de los sistemas de bases de datos

17

La arquitectura de un sistema de bases de datos está influida en gran medida por el sistema informático subyacente en el que se ejecuta, en concreto por aspectos de la arquitectura de la computadora como la conexión en red, el paralelismo y la distribución:

- La conexión en red de varias computadoras permite que algunas tareas se ejecuten en un sistema servidor y que otras lo hagan en los sistemas cliente. Esta división del trabajo ha conducido al desarrollo de *sistemas de bases de datos cliente-servidor*.
- El procesamiento paralelo en una computadora permite acelerar las actividades del sistema de base de datos, proporcionando a las transacciones respuestas más rápidas, así como capacidad de ejecutar más transacciones por segundo. Las consultas pueden procesarse de manera que se explote el paralelismo ofrecido por el sistema informático subyacente. La necesidad del procesamiento paralelo de consultas ha conducido al desarrollo de los *sistemas de bases de datos paralelos*.
- La distribución de datos a través de las distintas sedes de una organización permite que estos datos residan donde han sido generados o donde son más necesarios, pero continuar siendo accesibles desde otros lugares o departamentos diferentes. Mantener varias copias de la base de datos en diferentes sitios permite que puedan continuar las operaciones sobre la base de datos aunque algún sitio se vea afectado por desastres naturales como inundaciones, incendios o terremotos. Los *sistemas distribuidos de bases de datos* manejan datos distribuidos geográfica o administrativamente a lo largo de múltiples sistemas de bases de datos.

En este capítulo se estudia la arquitectura de los sistemas de bases de datos comenzando con los tradicionales sistemas centralizados y tratando, más adelante, los sistemas de bases de datos cliente-servidor, paralelos y distribuidos.

17.1. Arquitecturas centralizadas y cliente-servidor

Los sistemas de bases de datos centralizados son aquellos que se ejecutan en un único sistema informático sin interacción con otros sistemas. Comprenden desde los sistemas de bases de datos monousuario que se ejecutan en computadoras personales, hasta los sistemas de bases de datos de alto rendimiento que se ejecutan en grandes sistemas. Por otro lado, los sistemas cliente-servidor dividen su funcionalidad entre un sistema servidor y múltiples sistemas clientes.

17.1.1. Sistemas centralizados

Una computadora moderna de propósito general consta de una a unas pocas unidades centrales de procesamiento y un cierto número de controladores para los dispositivos que se encuentran conectados a través de un bus común, el cual proporciona acceso a la memoria compartida (Figura 17.1). Los procesadores poseen memorias caché locales en las que se almacenan copias de ciertas partes de la memoria para acelerar el acceso a los datos. Cada procesador puede tener varios **núcleos** independientes, cada uno de los cuales puede ejecutar instrucciones en un flujo independiente. Cada controlador de dispositivo se encarga de un tipo específico de dispositivos (por ejemplo, una unidad de disco, una tarjeta de sonido o un monitor). Los procesadores y los controladores de dispositivos pueden ejecutarse concurrentemente compitiendo así por el acceso a la memoria. La memoria caché reduce la disputa por el acceso a la memoria, ya que disminuye el número de veces que el procesador necesita acceder a la memoria.

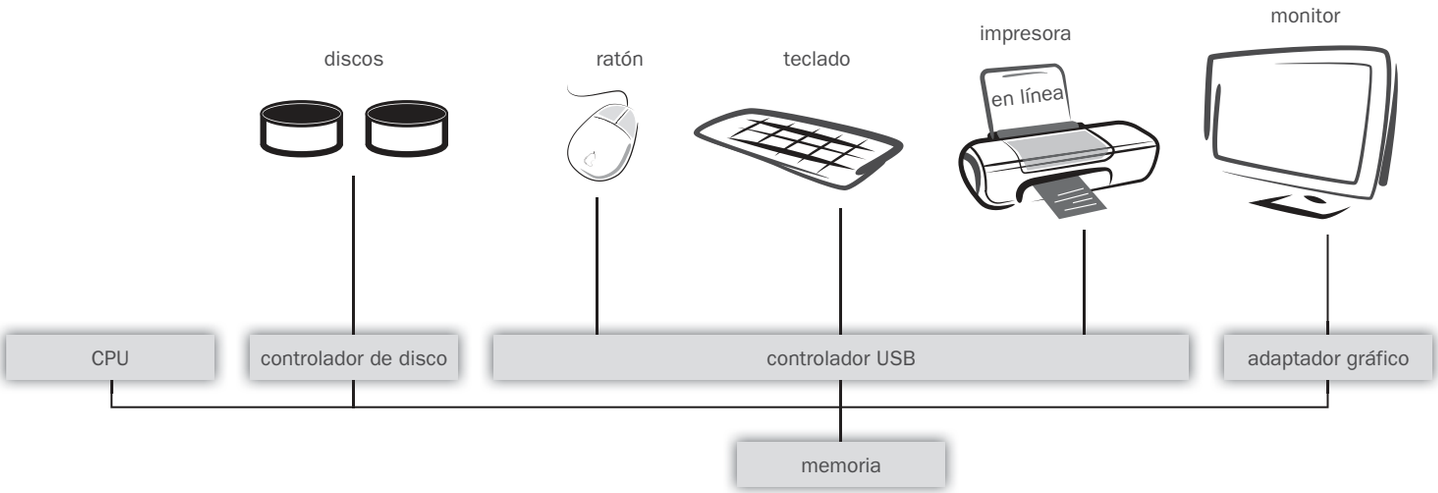


Figura 17.1. Un sistema centralizado.

Se distinguen dos formas de utilizar las computadoras: como sistemas monousuario o multiusuario. En la primera categoría se encuentran las computadoras personales y las estaciones de trabajo. Un **sistema monousuario** típico es una unidad de sobremesa utilizada por una única persona que dispone de un solo procesador, con uno o dos discos fijos, y que únicamente utiliza una persona a la vez. Por otra parte, un **sistema multiusuario** típico tiene más discos y más memoria y puede disponer de varios procesadores. Da servicio a un gran número de usuarios que están conectados al sistema de forma remota.

Normalmente, los sistemas de bases de datos diseñados para funcionar sobre sistemas monousuario no suelen proporcionar muchas de las facilidades que ofrecen los sistemas multiusuario. En particular, no tienen control de concurrencia, ya que no es necesario cuando solamente existe un usuario que puede generar modificaciones. Las facilidades de recuperación en estos sistemas o no existen o son primitivas; por ejemplo, realizar una copia de seguridad de la base de datos antes de cualquier modificación. La mayoría de estos sistemas no admiten SQL y proporcionan un lenguaje de consulta muy simple que, en algunos casos, es una variante de QBE. En cambio, los sistemas de bases de datos diseñados para sistemas multiusuario soportan todas las características de las transacciones que se han estudiado anteriormente.

Aunque en la actualidad las computadoras de propósito general disponen de varios procesadores, utilizan **paralelismo de grano grueso,** con solo unos pocos procesadores (normalmente dos o cuatro) que comparten la misma memoria principal. Las bases de datos que se ejecutan en tales máquinas habitualmente no intentan dividir una consulta simple entre los distintos procesadores, sino que ejecutan cada consulta en un único procesador posibilitando la concurrencia de varias consultas. Así, estos sistemas consiguen un mayor rendimiento, es decir, permiten ejecutar un mayor número de transacciones por segundo, a pesar de que cada transacción individualmente no se ejecute más rápido.

Las bases de datos diseñadas para las máquinas monoprocesador ya disponen de multitarea, permitiendo que varios procesos se ejecuten a la vez en el mismo procesador, usando tiempo compartido, ofreciendo de cara al usuario procesos que se están ejecutando en paralelo. De esta manera, desde un punto de vista lógico, las máquinas paralelas de grano grueso parecen ser idénticas a las máquinas monoprocesador, de forma que los sistemas de bases de datos diseñados para máquinas de tiempo compartido se pueden adaptar fácilmente para que puedan ejecutarse sobre máquinas paralelas de grano grueso.

Por otra parte, las **máquinas paralelas de grano fino** disponen de un gran número de procesadores y los sistemas de bases de datos que se ejecutan sobre ellas intentan ejecutar con paralelismo las tareas simples (consultas, por ejemplo) que solicitan los usuarios. En la Sección 17.3 se estudia la arquitectura de los sistemas de bases de datos paralelos.

El paralelismo está surgiendo como uno de los elementos críticos en el futuro diseño de los sistemas de bases de datos. Mientras que en la actualidad los sistemas de computadoras con procesadores multinúcleo disponen de unos pocos núcleos, en el futuro los procesadores dispondrán de un gran número de ellos.[1] Por ello, los sistemas de bases de datos paralelos, que eran sistemas especializados que se ejecutaban en un hardware específicamente diseñado, se convertirán en sistemas normales.

1 La razón para ello se basa en temas de arquitectura de computadoras relacionados con la generación de calor y el consumo de energía. En lugar de fabricar procesadores significativamente más rápidos, los arquitectos de computadoras usan los avances en diseño de chips para incluir más núcleos en un único chip, una tendencia que probablemente continúe durante algún tiempo.

17.1.2. Sistemas cliente-servidor

A medida que las computadoras personales se han ido haciendo cada vez más rápidas, más potentes y más baratas, los sistemas se han ido distanciando de la arquitectura centralizada. Los terminales conectados a un sistema central han sido suplantados por computadoras personales. De igual forma, la interfaz de usuario, que solía estar gestionada directamente por el sistema central, está pasando a ser gestionada, cada vez más, por las computadoras personales. Como consecuencia, los sistemas centralizados actúan hoy como **sistemas servidores** que satisfacen las peticiones generadas por los *sistemas clientes*. En la Figura 17.2 se representa la estructura general de un sistema cliente-servidor.

La funcionalidad que proporcionan los sistemas de bases de datos se puede dividir a grandes rasgos en dos partes: la fachada (*front-end*) y el sistema subyacente (*back-end*). El sistema subyacente gestiona el acceso a las estructuras, la evaluación y optimización de consultas, el control de concurrencia y la recuperación. La fachada de un sistema de base de datos está formada por herramientas como la interfaz de usuario con SQL, interfaces de formularios, diseñadores de informes y herramientas para la recopilación y análisis de datos (véase la Figura 17.3). La interfaz entre la fachada y el sistema subyacente puede realizarse con SQL o con una aplicación.

Las normas como *ODBC* y *JDBC*, que se estudiaron en el Capítulo 3, se desarrollaron para hacer de interfaz entre clientes y servidores. Cualquier cliente que utilice interfaces ODBC o JDBC puede conectarse a cualquier servidor que proporcione la interfaz.

Ciertas aplicaciones, como las hojas de cálculo y los paquetes de análisis estadístico, utilizan directamente la interfaz cliente-servidor para acceder a los datos del servidor subyacente. De hecho, proporcionan interfaces visibles especiales para diferentes tareas.

Los sistemas que trabajan con una gran cantidad de usuarios adoptan una arquitectura de tres capas, que se vio anteriormente en la Figura 1.6 (Capítulo 1), en la que la fachada es el navegador web que se comunica con un servidor de aplicaciones. El servidor de aplicaciones actúa como cliente del servidor de bases de datos.

Algunos sistemas de procesamiento de transacciones proporcionan una interfaz de **llamada a procedimientos remotos para transacciones** para conectar a los clientes con el servidor. Estas llamadas aparecen para el programador como llamadas normales a procedimientos, pero todas las llamadas a procedimientos remotos hechas desde un cliente se engloban en una única transacción al servidor final. De este modo, si la transacción se cancela, el servidor puede deshacer los efectos de las llamadas a procedimientos remotos individuales.

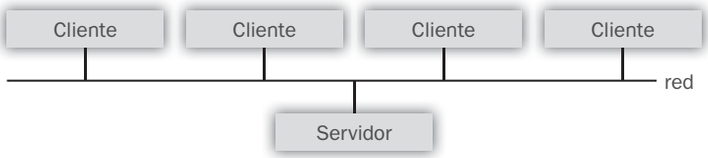


Figura 17.2. Estructura general de un sistema cliente–servidor.

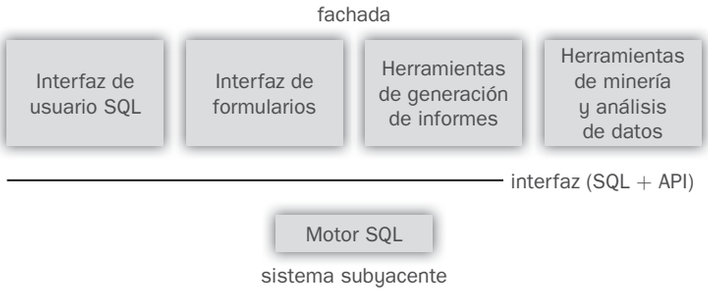


Figura 17.3. Funcionalidades de la fachada y del sistema subyacente.

17.2. Arquitecturas de sistemas servidores

Los sistemas servidores pueden dividirse en servidores de transacciones y servidores de datos.

- Los sistemas **servidores de transacciones,** también llamados sistemas **servidores de consultas,** proporcionan una interfaz por la que los clientes pueden enviar peticiones para realizar una acción que el servidor ejecutará y cuyos resultados se devolverán al cliente. Normalmente, las máquinas cliente envían las transacciones a los sistemas servidores, lugar en el que estas transacciones se ejecutan, y los resultados se devuelven a los clientes que son los encargados de visualizar los datos. Las peticiones se pueden especificar utilizando SQL o mediante la interfaz de una aplicación especializada.
- Los **sistemas servidores de datos** permiten a los clientes interaccionar con los servidores realizando peticiones de lectura o modificación de datos en unidades tales como archivos o páginas. Por ejemplo, los servidores de archivos proporcionan una interfaz de sistema de archivos a través de la cual los clientes pueden crear, modificar, leer y borrar archivos. Los servidores de datos de los sistemas de bases de datos ofrecen muchas más funcionalidades; soportan unidades de datos de menor tamaño que los archivos (como páginas, tuplas u objetos). Proporcionan facilidades de indexación de los datos, así como facilidades de transacción, de modo que los datos nunca se quedan en un estado inconsistente si falla una máquina cliente o un proceso.

De estas, la arquitectura del servidor de transacciones es, con mucho, la arquitectura más ampliamente utilizada. En las Secciones 17.2.1 y 17.2.2 se desarrollarán las arquitecturas de los servidores de transacciones y de los servidores de datos.

17.2.1. Servidores de transacciones

Un sistema servidor de transacciones típico consta de múltiples procesos accediendo a los datos en una memoria compartida, como en la Figura 17.4. Los procesos que forman parte del sistema de bases de datos incluyen:

- **Procesos de servidor.** Son procesos que reciben consultas del usuario (transacciones), las ejecutan, y devuelven los resultados. Las consultas pueden enviarse a los procesos de servidor desde la interfaz de usuario, o desde un proceso de usuario que ejecute SQL incorporado, o a través de JDBC, ODBC o protocolos similares. Algunos sistemas de bases de datos utilizan un proceso distinto para cada sesión de usuario, y unos pocos utilizan un único proceso de la base de datos para todas las sesiones del usuario, pero con múltiples hebras o hilos,[2] de forma que se pueden ejecutar concurrentemente múltiples consultas. (Un **hilo** es parecido a un proceso; sin embargo, varios hilos se pueden ejecutar concurrentemente como parte de un mismo proceso, y todos ellos en el mismo espacio de memoria virtual). Muchos sistemas de bases de datos utilizan una arquitectura híbrida, con procesos múltiples, cada uno de ellos con varios hilos.
- **Proceso gestor de bloqueos.** Este proceso implementa una función de gestión de bloqueos que incluye concesión de bloqueos, liberación de bloqueos y detección de interbloqueos.
- **Proceso escritor de bases de datos.** Existe uno o más procesos que vuelcan al disco los bloques de memoria intermedia modificados de forma continua.
- **Proceso escritor del registro histórico.** Este proceso genera entradas de registro en el almacenamiento estable a partir de la memoria intermedia del registro histórico. Los procesos de servidor simplifican la adición de entradas a la memoria intermedia del registro en memoria compartida y si es necesario forzar la escritura del registro, le piden al proceso escritor de registros que vuelque las entradas del registro.
- **Proceso punto de revisión.** Este proceso realiza periódicamente puntos de revisión.
- **Proceso monitor de procesos.** Este proceso observa otros procesos y, si cualquiera de ellos falla, realiza acciones de recuperación para el proceso, tales como cancelar cualquier transacción que estuviera ejecutando el proceso fallido, y reinicia el proceso.

La memoria compartida contiene todos los datos compartidos, como:

- Grupo de memorias intermedias.
- Tabla de bloqueos.
- Memoria intermedia del registro, que contiene las entradas del registro que esperan a ser volcadas en el almacenamiento estable.
- Planes de consulta en caché, que se pueden reutilizar si se envía de nuevo la misma consulta.

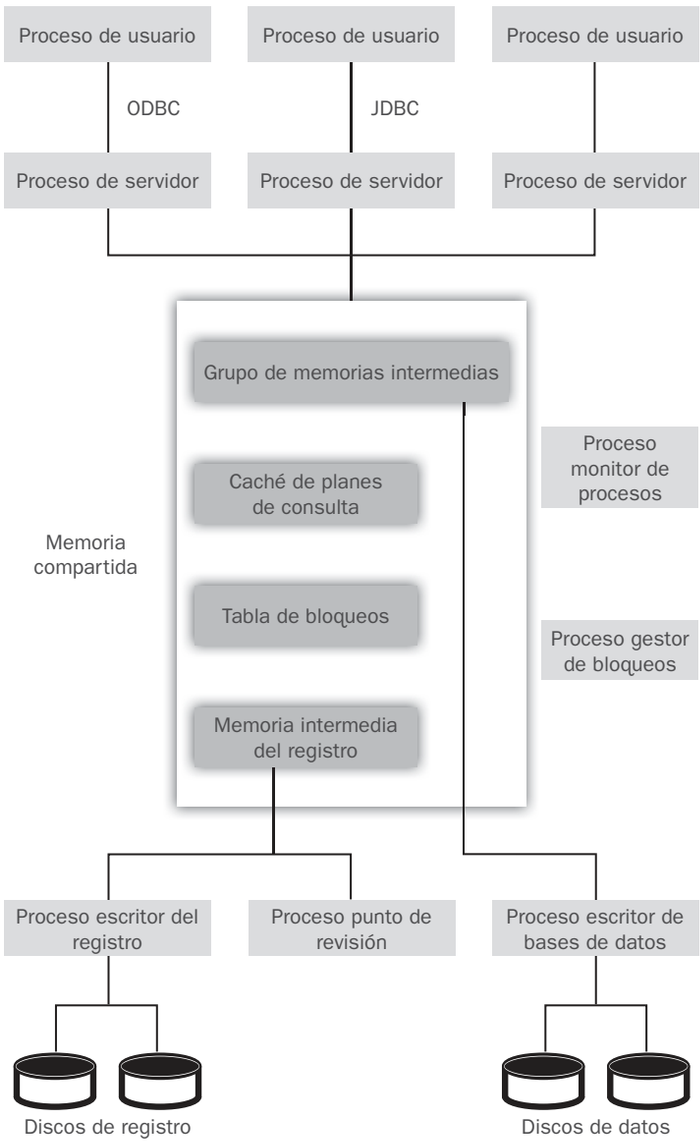


Figura 17.4. Estructura de la memoria compartida y de los procesos.

2 A lo largo del texto se encontrará hilo o hebra para referirse al mismo concepto. (*N. del E.*)

Todos los procesos de la base de datos pueden acceder a los datos de la memoria compartida. Como hay múltiples procesos, que pueden leer o realizar actualizaciones en las estructuras de datos en memoria compartida, debe haber un mecanismo que asegure que solo uno de ellos está modificando una estructura de datos en un momento dado, y que ningún proceso está leyendo una estructura de datos mientras otros la escriben. Tal **exclusión mutua** se puede implementar por medio de funciones del sistema operativo llamadas semáforos. Implementaciones alternativas, con menos sobrecargas, utilizan **instrucciones atómicas** especiales soportadas por el hardware de la computadora; un tipo de instrucción atómica comprueba una posición de la memoria y la establece a uno automáticamente. Se pueden encontrar más detalles sobre la excusión mutua en cualquier libro de texto estándar sobre sistemas operativos. Los mecanismos de exclusión mutua también se utilizan para implementar pestillos.

Para evitar la sobrecarga del paso de mensajes, en muchos sistemas de bases de datos los procesos servidor implementan el bloqueo actualizando directamente la tabla de bloqueos (que está en memoria compartida), en lugar de enviar mensajes de solicitud de bloqueo a un proceso administrador de bloqueos. El procedimiento de solicitud de bloqueos ejecuta las acciones que realizaría el proceso administrador de bloqueos para procesar una solicitud. Las acciones de la solicitud y la liberación de bloqueos son como las de la Sección 15.1.4, pero con dos diferencias significativas:

- Dado que varios procesos servidor pueden acceder a la memoria compartida, se debe asegurar la exclusión mutua en la tabla de bloqueos.
- Si no se puede obtener un bloqueo inmediatamente a causa de un conflicto de bloqueos, el código de la solicitud de bloqueo sigue observando la tabla de bloqueos hasta percatarse de que se ha concedido. El código de liberación de bloqueo actualiza la tabla de bloqueos para indicar a qué proceso se le ha concedido el bloqueo.

  Para evitar repetidas comprobaciones de la tabla de bloqueos, el código de solicitud de bloqueo puede utilizar los semáforos del sistema operativo para esperar una notificación de una concesión. El código de liberación de bloqueo debe utilizar entonces el mecanismo de semáforos para notificar a las transacciones que están esperando que sus bloqueos han sido concedidos.

Incluso si el sistema gestiona las solicitudes de bloqueo por medio de memoria compartida, sigue utilizando el proceso administrador de bloqueos para la detección de interbloqueos.

17.2.2. Servidores de datos

Los sistemas servidores de datos se utilizan en redes de área local en las que se alcanza una alta velocidad de conexión entre los clientes y el servidor. Las máquinas clientes son comparables al servidor en cuanto a potencia de procesamiento, y ejecutan tareas de cómputo intensivo. En este entorno tiene sentido enviar los datos a las máquinas clientes, realizar allí todo el procesamiento (que puede durar un tiempo) y después enviar los datos de vuelta al servidor. Observe que esta arquitectura necesita que los clientes posean todas las funcionalidades del sistema subyacente. Las arquitecturas de los servidores de datos se han hecho particularmente populares en los sistemas de bases de datos orientadas a objetos (Capítulo 22).

En esta arquitectura surgen algunos aspectos interesantes, ya que el coste en tiempo de comunicación entre el cliente y el servidor es alto, comparado con el de acceso a una memoria local (milisegundos frente a menos de 100 nanosegundos).

- **Envío de páginas** frente al **envío de elementos.** La unidad de comunicación de datos puede ser de grano grueso, como una página, o de grano fino, como una tupla (o, en el contexto de los sistemas de bases de datos orientados a objetos, un objeto). Se empleará el término **elemento** para referirse tanto a tuplas como a objetos.

  Si la unidad de comunicación de datos es un único elemento, la sobrecarga por la transferencia de mensajes es alta, comparada con el número de datos transmitidos. En su lugar, cuando se necesita un elemento, cobra sentido la idea de enviar junto a él otros que probablemente vayan a emplearse en un futuro próximo. Se denomina **preextracción** a la acción de buscar y enviar elementos antes de que sea estrictamente necesario. Si varios elementos residen en un página, el envío de páginas puede considerarse como una forma de preextracción ya que, cuando un proceso desee acceder a un único elemento de la página, se enviarán todos los elementos de esa página.
- **Granularidad adaptativa de bloqueo.** La concesión del bloqueo de los elementos de datos que el servidor envía a los clientes la realiza habitualmente el propio servidor. Un inconveniente del envío de páginas es que los clientes pueden recibir bloqueos de grano grueso; el bloqueo de una página bloquea implícitamente todos los elementos que residen en ella. El cliente adquiere implícitamente bloqueos sobre todos los elementos preextraídos incluso aunque no esté accediendo a algunos de ellos. De esta forma, es posible que se detenga innecesariamente el procesamiento de otros clientes que necesiten bloquear estos elementos. Se han propuesto algunas técnicas para la **liberación** de bloqueos en las que el servidor puede pedir a los clientes que le devuelvan el control sobre los bloqueos de los elementos preextraídos. Si el cliente no necesita el elemento preextraído puede devolver los bloqueos sobre ese elemento al servidor para que estos puedan ser asignados a otros clientes.
- **Caché de datos.** Los datos que se envían al cliente en favor de una transacción se pueden **alojar en una caché** del cliente incluso una vez completada la transacción, si dispone de suficiente espacio de almacenamiento libre. Las transacciones sucesivas en el mismo cliente pueden hacer uso de los datos en caché. Sin embargo, se presenta el problema de la **coherencia de caché:** si una transacción encuentra los datos en la caché, debe asegurarse de que esos datos están actualizados ya que, después de haber sido almacenados en la caché, pueden haber sido modificados por otro cliente. Así, debe establecerse una comunicación con el servidor para comprobar la validez de los datos y poder adquirir un bloqueo sobre ellos.
- **Caché de bloqueos.** Los bloqueos también se pueden almacenar en la memoria caché del cliente si los datos están prácticamente divididos entre los clientes, de manera que un cliente rara vez necesite los datos de otros clientes. Suponga que se encuentran en la memoria caché tanto el elemento de datos que se busca como el bloqueo requerido para acceder al mismo. Entonces, el cliente puede acceder al elemento de datos sin necesidad de comunicar nada al servidor. No obstante, el servidor debe seguir el rastro de los bloqueos en caché; si un cliente solicita un bloqueo al servidor, este debe **retrollamar** a todos los bloqueos sobre el elemento de datos que se encuentren en las memorias caché de otros clientes. La tarea se vuelve más complicada cuando se tienen en cuenta los posibles fallos de la máquina. Esta técnica se diferencia de la liberación de bloqueos en que la caché de bloqueo se realiza a través de transacciones; de otra forma, las dos técnicas serían similares.

Las notas bibliográficas proporcionan más información sobre los sistemas cliente-servidor de bases de datos.

17.2.3. Servidores en la nube

Los servidores suelen ser propiedad de la empresa que proporciona el servicio, pero existe una tendencia creciente de proveedores de servicios que confían, al menos en parte, que sus servidores sean propiedad de «un tercero» que no es el cliente ni el proveedor de servicios.

Un modelo para usar servidores de terceros es subcontratar todo el servicio a otra compañía que tiene el servicio completo en sus propias computadoras usando su propio software. De esta forma el proveedor de servicio ignora la mayor parte de los detalles de la tecnología y se centra en la venta del servicio.

Otro modelo para el uso de servidores de terceros es la **computación en la nube,** en la que los proveedores de servicios ejecutan su propio software, pero lo ejecutan en computadoras que proporciona otra compañía. Con este modelo, la tercera parte no proporciona ninguna aplicación software; únicamente proporciona una colección de máquinas. Estas máquinas no son máquinas «reales», sino simuladas por un software que permite que una única computadora real simule varias computadoras independientes. Estas máquinas simuladas se llaman **máquinas virtuales.** El proveedor del servicio ejecuta su software (posiblemente incluso un sistema de bases de datos) en estas máquinas virtuales. Una gran ventaja de la computación en la nube es que el proveedor del servicio puede añadir máquinas si fuesen necesarias para atender la demanda y liberarlas en momentos de poca carga. Esto la hace ser muy efectiva en costes tanto en términos de dinero como de energía.

Un tercer modelo usa el servicio de computación en la nube como un servidor de datos, como los sistemas de *almacenamiento de datos en la nube;* sistemas que se tratan con detalle en la Sección 19.9. Las aplicaciones de bases de datos que usen almacenamiento de datos en la nube pueden ejecutar en la misma nube (es decir, en el mismo conjunto de máquinas) o en otra nube. Las notas bibliográficas proporcionan más información sobre los sistemas de computación en la nube.

17.3. Sistemas paralelos

Los sistemas paralelos mejoran la velocidad de procesamiento y de E/S usando varios procesadores y discos en paralelo. Cada vez son más comunes las máquinas paralelas, lo que hace que cada vez sea más importante el estudio de los sistemas paralelos de bases de datos. La fuerza que ha impulsado a los sistemas paralelos de bases de datos ha sido la demanda de aplicaciones que han de manejar bases de datos extremadamente grandes (del orden de terabytes; es decir, 10^{12} bytes) o que tienen que procesar un número enorme de transacciones por segundo (del orden de miles de transacciones por segundo). Los sistemas de bases de datos centralizados o cliente-servidor no son suficientemente potentes para estas aplicaciones.

En el procesamiento paralelo se realizan muchas operaciones simultáneamente, mientras que en el procesamiento secuencial los distintos pasos computacionales han de ejecutarse en serie. Una máquina paralela de **grano grueso** consta de un pequeño número de potentes procesadores; una máquina **masivamente paralela** o de **grano fino** utiliza miles de procesadores más pequeños. Hoy en día, la mayoría de las máquinas de gama alta ofrecen un cierto grado de paralelismo de grano grueso: son comunes las máquinas con dos o cuatro procesadores. Las computadoras masivamente paralelas se distinguen de las máquinas paralelas de grano grueso porque admiten un grado de paralelismo mucho mayor. Ya se encuentran comercialmente computadoras paralelas con cientos de procesadores y discos.

Para evaluar el rendimiento de los sistemas de bases de datos existen dos medidas principales: (1) el **rendimiento,** es decir, el número de tareas que pueden completarse en un intervalo de tiempo determinado, y (2) el **tiempo de respuesta,** el tiempo necesario para completar una única tarea a partir del momento en que se envíe. Un sistema que procese un gran número de pequeñas transacciones puede mejorar el rendimiento realizando muchas transacciones en paralelo. Un sistema que procese transacciones largas puede mejorar el tiempo de respuesta y la productividad realizando en paralelo las distintas subtareas de cada transacción.

17.3.1. Ganancia de velocidad y ampliabilidad

La ganancia de velocidad y la ampliabilidad son dos aspectos importantes en el estudio del paralelismo. La **ganancia de velocidad** se refiere a la ejecución en menos tiempo de una tarea dada mediante el incremento del grado de paralelismo. La **ampliabilidad** hace referencia al manejo de transacciones más largas mediante el incremento del grado de paralelismo.

Considere una aplicación de base de datos ejecutándose en un sistema paralelo con un cierto número de procesadores y discos. Suponga ahora que se incrementa el tamaño del sistema añadiéndole más procesadores, discos y otros componentes. El objetivo es realizar el procesamiento de la tarea en un tiempo inversamente proporcional al número de procesadores y discos del sistema. Suponga que el tiempo de ejecución de una tarea en la máquina más grande es T_G y que el tiempo de ejecución de la misma tarea en la máquina más pequeña es T_P. La ganancia de velocidad debida al paralelismo se define como T_P / T_G. Se dice que un sistema paralelo tiene una **ganancia de velocidad lineal** si la ganancia de velocidad es N cuando el sistema más grande tiene N veces más recursos (procesadores, discos, etc.) que el sistema más pequeño. Si la ganancia de velocidad es menor que N se dice que el sistema tiene una **ganancia de velocidad sublineal.** En la Figura 17.5 se muestra la ganancia de velocidad lineal y sublineal.

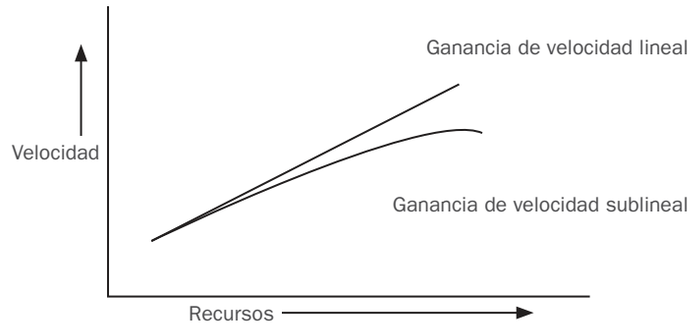


Figura 17.5. Ganancia de velocidad respecto al incremento de los recursos.

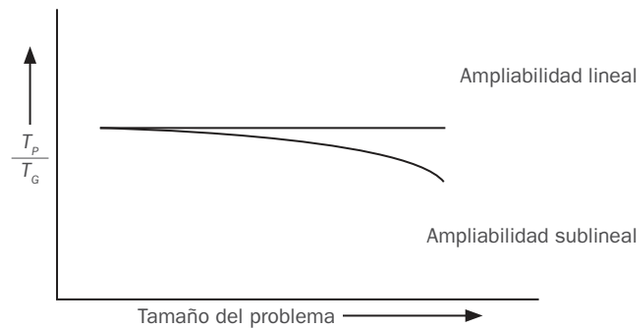


Figura 17.6. Ampliabilidad respecto al crecimiento del tamaño del problema y de los recursos.

La ampliabilidad está relacionada con la capacidad para procesar tareas más largas en el mismo tiempo mediante el incremento de los recursos del sistema. Sea Q una tarea y sea Q_N una tarea N veces más grande que Q. Suponga que T_P es el tiempo de ejecución de la tarea Q en una máquina dada M_P y que T_G es el tiempo de ejecución de la tarea Q_N en una máquina paralela M_G, que es N veces más grande que M_P. La ampliabilidad se define como T_P / T_G. Se dice que el sistema paralelo M_G tiene una **ampliabilidad lineal** sobre la tarea Q si $T_G = T_P$. Si $T_G > T_P$ se dice que el sistema tiene una **ampliabilidad sublineal.** En la Figura 17.6 se muestra la ampliabilidad lineal y sublineal (en la que los recursos aumentan proporcionalmente al tamaño del problema). La manera de medir el tamaño de las tareas da lugar a dos tipos de ampliabilidad relevantes en los sistemas paralelos de bases de datos:

- En la **ampliabilidad por lotes,** el tamaño de la base de datos aumenta, y las tareas son trabajos grandes cuyo tiempo de ejecución depende del tamaño de la base de datos. Un ejemplo de dicha tarea puede ser explorar una relación cuyo tamaño es proporcional a la base de datos. Por tanto, el tamaño de la base de datos es una medida del tamaño del problema. La ampliabilidad por lotes también se aplica en aplicaciones científicas, como la ejecución de una consulta con una resolución N veces más fina o una simulación N veces más larga.
- En la **ampliabilidad de transacciones,** la velocidad a la que se envían las transacciones a la base de datos aumenta con el tamaño de la misma de forma proporcional a la tasa de las transacciones. Este tipo de ampliabilidad es relevante en los sistemas de procesamiento de transacciones en los que las transacciones son pequeñas actualizaciones (por ejemplo, un depósito o una retirada de fondos de una cuenta) y la tasa de transacciones crece según se crean más cuentas. Este tipo de transacción está especialmente adaptada para la ejecución paralela, ya que las transacciones se pueden ejecutar concurrente e independientemente en procesadores separados, y cada transacción tarda aproximadamente el mismo tiempo, incluso si la base de datos crece.

La ampliabilidad es normalmente la métrica más importante para medir la eficiencia de un sistema paralelo de bases de datos. El objetivo del paralelismo en los sistemas de bases de datos suele ser asegurar que la ejecución del sistema continuará realizándose a una velocidad aceptable, incluso en el caso de que aumente el tamaño de la base de datos o el número de transacciones. El incremento de la capacidad del sistema mediante el aumento del paralelismo proporciona a una empresa un modo de crecimiento más suave que el de reemplazar un sistema centralizado por una máquina más rápida (suponiendo incluso que esta máquina existiera). Sin embargo, cuando se utiliza la ampliabilidad debe atenderse también a los valores del rendimiento absoluto; una máquina con un ampliabilidad lineal puede tener un rendimiento más bajo que otra con ampliabilidad sublineal simplemente porque la última sea mucho más rápida inicialmente.

Existen algunos factores que trabajan en contra de la eficiencia del paralelismo y pueden atenuar tanto la ganancia de velocidad como la ampliabilidad.

- **Costes de inicio.** El inicio de un único proceso lleva asociado un coste. En una operación paralela compuesta por miles de procesos el *tiempo de inicio* puede llegar a ser mucho mayor que el tiempo real de procesamiento, lo que influye negativamente en la ganancia de velocidad.
- **Interferencia.** Como los procesos que se ejecutan en un sistema paralelo acceden con frecuencia a recursos compartidos, pueden sufrir un cierto retardo como consecuencia de la *interferencia* de cada nuevo proceso en la competencia con los procesos existentes por el acceso a los recursos más comunes, como el bus del sistema, los discos compartidos o incluso los bloqueos. Este fenómeno afecta tanto a la ganancia de velocidad como a la ampliabilidad.
- **Sesgo.** Al dividir cada tarea en un cierto número de pasos paralelos se reduce el tamaño del paso medio. Es más, el tiempo de servicio de la tarea completa vendrá determinado por el tiempo de servicio del paso más lento. Normalmente es difícil dividir una tarea en partes exactamente iguales; entonces se dice que la forma de distribución de los tamaños es *sesgada*. Por ejemplo, si se divide una tarea de tamaño 100 en 10 partes y la división está sesgada, puede haber algunas tareas de tamaño menor que 10 y otras de tamaño superior a 10; si el tamaño de una tarea fuera 20, la ganancia de velocidad que se obtendría al ejecutar las tareas en paralelo solo valdría 5 en vez de lo que cabría esperarse, 10.

17.3.2. Redes de interconexión

Los sistemas paralelos están constituidos por un conjunto de componentes (procesadores, memoria y discos) que pueden comunicarse entre sí a través de una **red de interconexión.** La Figura 17.7 muestra tres tipos de redes de interconexión utilizados habitualmente:

- **Bus.** Todos los componentes del sistema pueden enviar o recibir datos de un único bus de comunicaciones. Este tipo de interconexión se muestra en la Figura 17.7a. El bus puede ser una red Ethernet o una interconexión paralela. Las arquitecturas de bus funcionan bien para un número pequeño de procesadores. Sin embargo, como el bus solo puede gestionar la comunicación de un único componente en cada momento, las arquitecturas de bus son menos apropiadas según aumenta el paralelismo.
- **Malla.** Los componentes se organizan como los nodos de una retícula de modo que cada componente está conectado con todos los nodos adyacentes. En una malla bidimensional cada nodo está conectado con cuatro nodos adyacentes, mientras que en una malla tridimensional cada nodo está conectado con seis nodos adyacentes. La Figura 17.7b muestra una malla bidimensional. Los nodos entre los que no existe una conexión directa pueden comunicarse mediante el envío de mensajes a través de una secuencia de nodos intermedios que sí dispongan de conexión directa. A medida que aumenta el número de componentes también se incrementa el número de enlaces de comunicación, por lo que la capacidad de comunicación de una malla es mejor cuanto mayor es el paralelismo.
- **Hipercubo.** Se asigna a cada componente un número binario de modo que dos componentes tienen una conexión directa si sus correspondientes representaciones binarias difieren en un solo bit. Así, cada uno de los n componentes está conectado con otros $\log(n)$ componentes. La Figura 17.7c muestra un hipercubo con ocho vértices. Puede demostrarse que, en un hipercubo, un mensaje de un componente puede llegar a cualquier otro componente de la red de interconexión atravesando a lo sumo $\log(n)$ enlaces. Por el contrario, en una malla un componente puede estar a $2(\sqrt{n} - 1)$ enlaces de otros componentes (o a $\sqrt{n}$ enlaces de distancia si la malla de interconexión conecta entre sí los bordes opuestos). Por tanto, el retardo de la comunicación en un hipercubo es significativamente menor que en una malla.

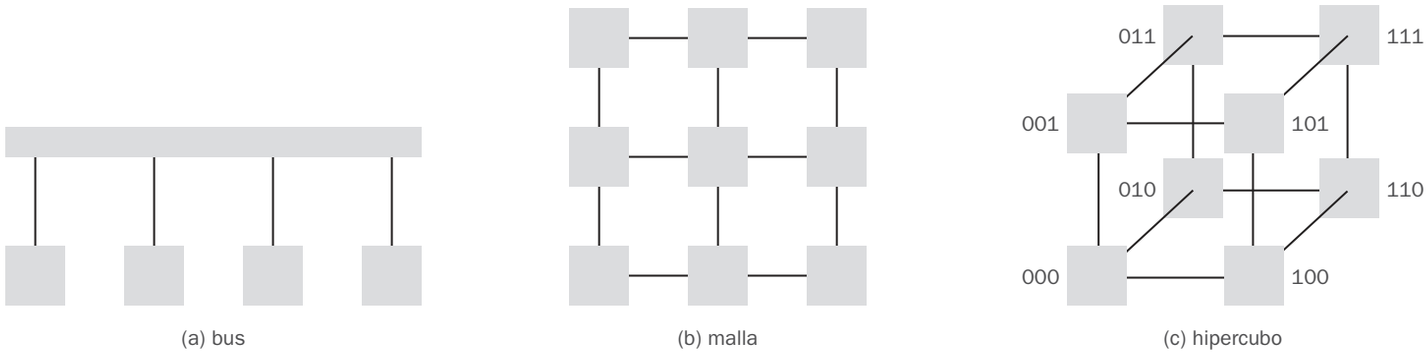


Figura 17.7. Interconexión de redes.

17.3.3. Arquitecturas paralelas de bases de datos

Existen varios modelos de arquitecturas para las máquinas paralelas. En la Figura 17.8 se muestran algunos de los más importantes (en la figura, M quiere decir memoria, P procesador y los discos se dibujan como cilindros):

- **Memoria compartida.** Todos los procesadores comparten una memoria común (Figura 17.8a).
- **Disco compartido.** Todos los procesadores comparten un conjunto de discos común (Figura 17.8b). Algunas veces los sistemas de disco compartido se denominan **agrupaciones.**
- **Sin compartimiento.** Los procesadores no comparten ni memoria ni disco (Figura 17.8c).
- **Jerárquica.** Este modelo es un híbrido de las arquitecturas anteriores (Figura 17.8d).

En las Secciones 17.3.3.1 hasta la 17.3.3.4 se abordará cada uno de estos modelos.

Las técnicas utilizadas para acelerar el procesamiento de transacciones en sistemas servidores de datos, como la caché de datos y bloqueos y la liberación de bloqueos, que se describen en la Sección 17.2.2, también se pueden utilizar en bases de datos paralelas de discos compartidos, además de en bases de datos paralelas sin compartimiento. De hecho, son muy importantes para el procesamiento eficiente de transacciones en tales sistemas.

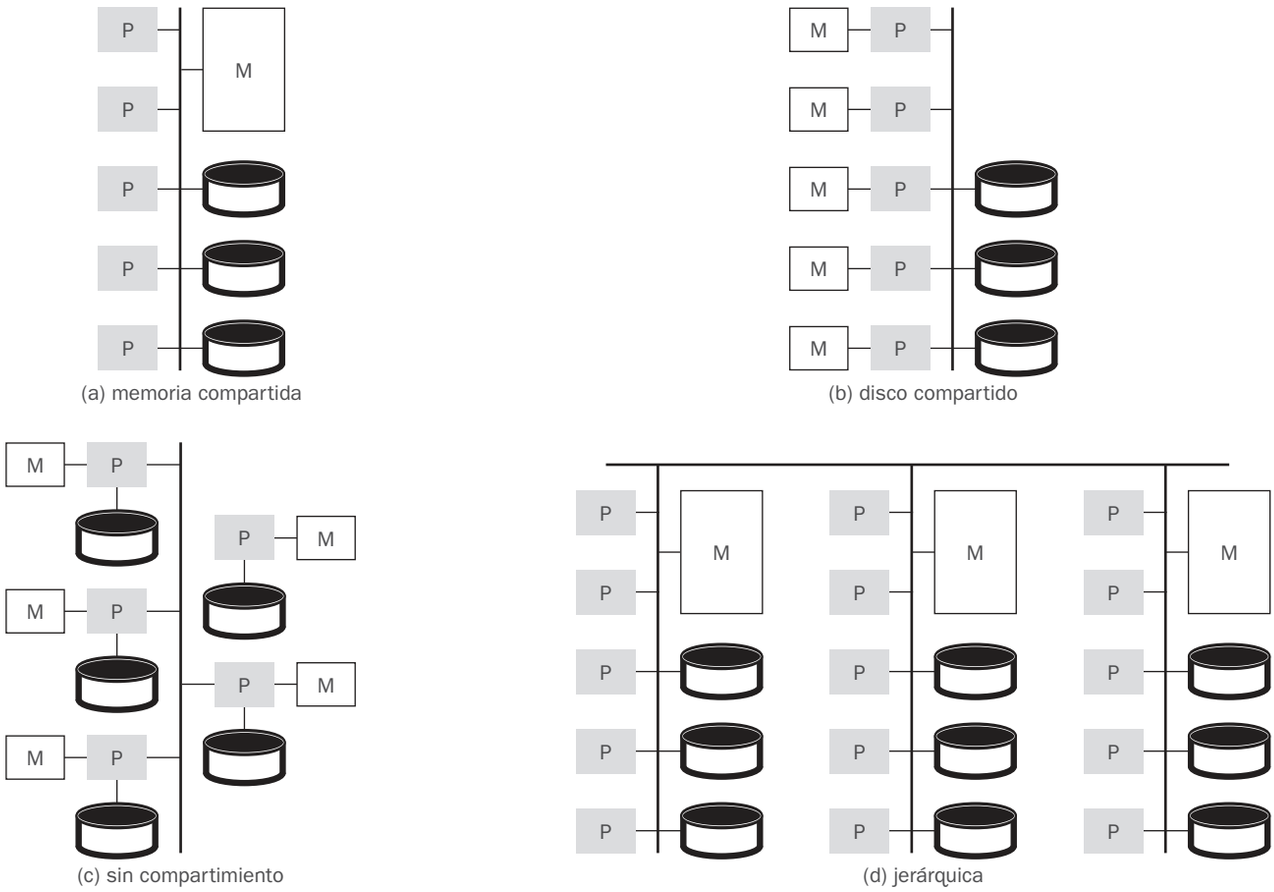


Figura 17.8. Arquitecturas paralelas de bases de datos.

17.3.3.1. Memoria compartida

En una arquitectura de **memoria compartida,** los procesadores y los discos tienen acceso a una memoria común, normalmente a través de un bus o de una red de interconexión. La ventaja de la memoria compartida es la extremada eficiencia en cuanto a la comunicación entre procesadores; cualquier procesador puede acceder a los datos de la memoria compartida sin que el software tenga que moverlos. Un procesador puede enviar mensajes a otros procesadores utilizando escrituras en la memoria de forma más rápida (normalmente inferior a un microsegundo) que enviando un mensaje con un mecanismo de comunicación. El inconveniente de las máquinas con memoria compartida es que la arquitectura no puede crecer más allá de 32 o 64 procesadores porque el bus o la red de interconexión se convertirían en un cuello de botella (ya que está compartido por todos los procesadores). Llega un momento en el que no sirve de nada añadir más procesadores, ya que estos emplean la mayoría de su tiempo esperando su turno para utilizar el bus y así poder acceder a la memoria.

Las arquitecturas de memoria compartida suelen dotar a cada procesador de una memoria caché muy grande para evitar las referencias a la memoria compartida siempre que sea posible. No obstante, en la caché no podrán estar todos los datos y no podrá evitarse el acceso a la memoria compartida. Además, las cachés necesitan mantener la coherencia; es decir, si un procesador realiza una escritura en una posición de memoria, los datos de dicha posición de memoria se deberían actualizar en, o eliminar de, cualquier procesador en el que los datos estuvieran en caché. El mantenimiento de la coherencia de la caché aumenta la sobrecarga cuando se incrementa el número de procesadores. Por estas razones las máquinas con memoria compartida no pueden extenderse llegado un punto; las máquinas actuales con memoria compartida no pueden soportar más de 64 procesadores.

17.3.3.2. Disco compartido

En el modelo de **disco compartido** todos los procesadores pueden acceder directamente a todos los discos a través de una red de interconexión, pero los procesadores tienen memorias privadas. Ofrecen dos ventajas respecto a las de memoria compartida. Primero, el bus de la memoria deja de ser un cuello de botella, ya que cada procesador dispone de memoria propia. Segundo, esta arquitectura ofrece una forma barata para proporcionar una cierta **tolerancia ante los fallos:** si falla un procesador (o su memoria), los demás procesadores pueden hacerse cargo de sus tareas ya que la base de datos reside en los discos, a los que tienen acceso todos los procesadores. Como se describía en el Capítulo 10, utilizando una arquitectura RAID también puede conseguirse que el subsistema de discos sea tolerante ante los fallos por sí mismo. La arquitectura de disco compartido tiene aceptación en bastantes aplicaciones.

El principal problema de los sistemas de discos compartidos es, de nuevo, la ampliabilidad. Aunque el bus de la memoria no es un cuello de botella muy grande, la interconexión con el subsistema de discos es ahora el nuevo cuello de botella; esto es especialmente grave en situaciones en las que la base de datos realiza un gran número de accesos a los discos. Los sistemas de discos compartidos pueden soportar un mayor número de procesadores en comparación con los sistemas de memoria compartida, pero la comunicación entre los procesadores es más lenta (hasta unos pocos milisegundos si se carece de un hardware de propósito especial para comunicaciones), ya que se realiza a través de una red de interconexión.

17.3.3.3. Sin compartimiento

En un sistema **sin compartimiento** cada nodo de la máquina consta de un procesador, la memoria y uno o más discos. Los procesadores de un nodo pueden comunicarse con un procesador de otro nodo utilizando una red de interconexión de alta velocidad. Un nodo funciona como el servidor de los datos almacenados en los discos que posee. El modelo sin compartimiento resuelve el inconveniente de requerir que todas las operaciones de E/S vayan a través de una única red de interconexión, ya que las referencias a los discos locales son servidas por los discos locales de cada procesador; solamente van por la red las peticiones, los accesos a discos remotos y las relaciones resultado. Es más, habitualmente las redes de interconexión para los sistemas sin compartimiento se diseñan para ser ampliables, por lo que su capacidad de transmisión crece a medida que se añaden nuevos nodos. Como consecuencia, las arquitecturas sin compartimiento son más ampliables y pueden admitir con facilidad un gran número de procesadores. El principal inconveniente de los sistemas sin compartimiento es el coste de comunicación y de acceso a discos remotos, coste que es mayor que el que se produce en las arquitecturas de memoria o disco compartido, ya que el envío de datos provoca la intervención del software en ambos extremos.

17.3.3.4. Jerárquica

La **arquitectura jerárquica** combina las características de las arquitecturas de memoria compartida, de disco compartido y sin compartimiento. A alto nivel, el sistema está formado por nodos que están conectados mediante una red de interconexión y que no comparten ni memoria ni discos. El nivel más alto es una arquitectura sin compartimiento. Cada nodo del sistema podría ser en realidad un sistema de memoria compartida con algunos procesadores. Alternativamente, cada nodo podría ser un sistema de disco compartido y cada uno de estos sistemas de disco compartido podría ser a su vez un sistema de memoria compartida. De esta manera, un sistema podría construirse como una jerarquía con una arquitectura de memoria compartida con pocos procesadores en la base, en lo más alto una arquitectura sin compartimiento y, quizá, una arquitectura de disco compartido en el medio. En la Figura 17.8d se muestra una arquitectura jerárquica con nodos de memoria compartida conectados entre sí con una arquitectura sin compartimiento. Hoy en día los sistemas paralelos comerciales de bases de datos pueden ejecutarse sobre varias de estas arquitecturas.

Los intentos por reducir la complejidad de programación de estos sistemas han dado lugar a las arquitecturas de **memoria virtual distribuida,** en las que hay una única memoria compartida desde el punto de vista lógico, pero hay varios sistemas de memoria disjuntos desde el punto de vista físico; se obtiene una única vista del área de memoria virtual de estas memorias disjuntas mediante un hardware de asignación de memoria virtual junto con el software del sistema. Como las velocidades de acceso son diferentes, dependiendo de si la página está disponible localmente o no, esta arquitectura también se denomina **arquitectura de memoria no uniforme** (NUMA: *Nonuniform Memory Architecture*).

17.4. Sistemas distribuidos

En un **sistema distribuido de bases de datos** se almacena la base de datos en varias computadoras. Los medios de comunicación, como las redes de alta velocidad o las líneas telefónicas, pueden poner en contacto las distintas computadoras de un sistema distribuido. No comparten ni memoria ni discos. Las computadoras de un sistema distribuido pueden variar en tamaño y función, pudiendo abarcar desde estaciones de trabajo a grandes sistemas.

Dependiendo del contexto en el que se mencionen, existen diferentes nombres para referirse a las computadoras que forman parte de un sistema distribuido, tales como **sitios** o **nodos.** Para enfatizar la distribución física de estos sistemas se emplea principalmente el término **sitio.** En la Figura 17.9 se muestra la estructura general de un sistema distribuido.

Las principales diferencias entre las bases de datos paralelas sin compartimiento y las bases de datos distribuidas son que las bases de datos distribuidas normalmente se encuentran en varios lugares geográficos distintos, se administran de forma separada y poseen una interconexión más lenta. Otra gran diferencia es que en un sistema distribuido existen dos tipos de transacciones: locales y globales. Una **transacción local** es aquella que accede a los datos del único sitio en el cual se inició la transacción. Por otra parte, una **transacción global** es la que, o bien accede a los datos situados en un sitio diferente de aquel en el que se inició la transacción, o bien accede a datos de varios sitios distintos.

Existen múltiples razones para construir sistemas distribuidos de bases de datos, incluyendo el compartimiento de los datos, la autonomía y la disponibilidad.

- **Datos compartidos.** La principal ventaja de construir un sistema distribuido de bases de datos es poder disponer de un entorno en el que los usuarios puedan acceder desde una única ubicación a los datos que residen en otras ubicaciones. Por ejemplo, en un sistema de una universidad distribuida, donde cada campus almacena los datos relacionados con dicho campus, es posible que un usuario de uno de los campus acceda a los datos de otro. Sin esta capacidad, un usuario que quisiera transferir registros de estudiantes de un campus a otro tendría que recurrir a algún mecanismo externo que pudiera enlazar los sistemas existentes.
- **Autonomía.** La principal ventaja de compartir datos mediante la distribución de datos es que cada ubicación es capaz de mantener un grado de control sobre los datos que se almacenan localmente. En un sistema centralizado, el administrador de bases de datos de la ubicación central controla la base de datos. En un sistema distribuido, existe un administrador de bases de datos global responsable de todo el sistema. Una parte de estas responsabilidades se delegan al administrador de bases de datos local de cada sitio. Dependiendo del diseño del sistema distribuido de bases de datos, cada administrador puede tener un grado diferente de **autonomía local.** La posibilidad de autonomía local es a menudo una de las grandes ventajas de las bases de datos distribuidas.
- **Disponibilidad.** Si un sitio de un sistema distribuido falla, los sitios restantes pueden seguir trabajando. En particular, si los elementos de datos están **replicados** en varios sitios, una transacción que necesite un elemento de datos en particular puede encontrarlo en varios sitios. De este modo, el fallo de un sitio no implica necesariamente la caída del sistema.

  El sistema debe detectar el fallo de un sitio y puede que sea necesario aplicar acciones apropiadas para la recuperación del fallo. El sistema no debe seguir utilizando los servicios del sitio que falló. Finalmente, cuando el sitio que falló se recupera o se repara, debe haber mecanismos disponibles para integrarlo sin problemas de nuevo en el sistema.

  Aunque la recuperación ante un fallo es más compleja en los sistemas distribuidos que en los sistemas centralizados, la capacidad que tienen muchos sistemas de continuar trabajando a pesar del fallo en uno de los sitios produce una mayor disponibilidad. La disponibilidad es crucial para los sistemas de bases de datos que se utilizan en aplicaciones de tiempo real. Que, por ejemplo, una línea aérea pierda el acceso a los datos puede provocar la pérdida de potenciales compradores de billetes en favor de la competencia.

17.4.1. Ejemplo de una base de datos distribuida

Considere un sistema bancario que consta de cuatro sucursales situadas en cuatro ciudades diferentes. Cada sucursal posee su propia computadora con una base de datos que guarda todas las cuentas abiertas en dicha sucursal. Así, cada una de estas instalaciones se considera un sitio. También hay un único sitio que mantiene la información relativa a todas las sucursales del banco.

Para mostrar la diferencia entre los dos tipos de transacciones (local y global) considere una transacción que añade 50 € a la cuenta A-177 situada en la sucursal de Valleyview. La transacción se considera local si esta comenzó en la sucursal de Valleyview; en otro caso, se considera global. Una transacción que transfiere 50 € desde la cuenta A-177 a la cuenta A-305, que se encuentra en la sucursal de Hillside, es una transacción global, ya que como resultado de su ejecución se accede a datos de dos sitios diferentes.

En un sistema distribuido de bases de datos ideal, los sitios deberían compartir un esquema global común (aunque algunas relaciones solo se puedan almacenar en algunos sitios), todos los sitios deberían ejecutar el mismo software de gestión de bases de datos distribuidas, y los sitios deberían conocer la existencia de los demás. Si una base de datos distribuida se construye partiendo de cero, realmente debería ser posible lograr los objetivos anteriores. Sin embargo, en la realidad, una base de datos distribuida se tiene que construir enlazando múltiples sistemas de bases de datos que ya existen, cada uno con su propio esquema y, posiblemente, ejecutando diferente software de gestión de bases de datos. A veces, tales sistemas reciben el nombre de **sistemas de bases de datos múltiples** o **sistemas de bases de datos distribuidos y heterogéneos.** En la Sección 19.8 se estudian dichos sistemas y se muestra cómo conseguir un cierto grado de control global a pesar de la heterogeneidad de los sistemas.

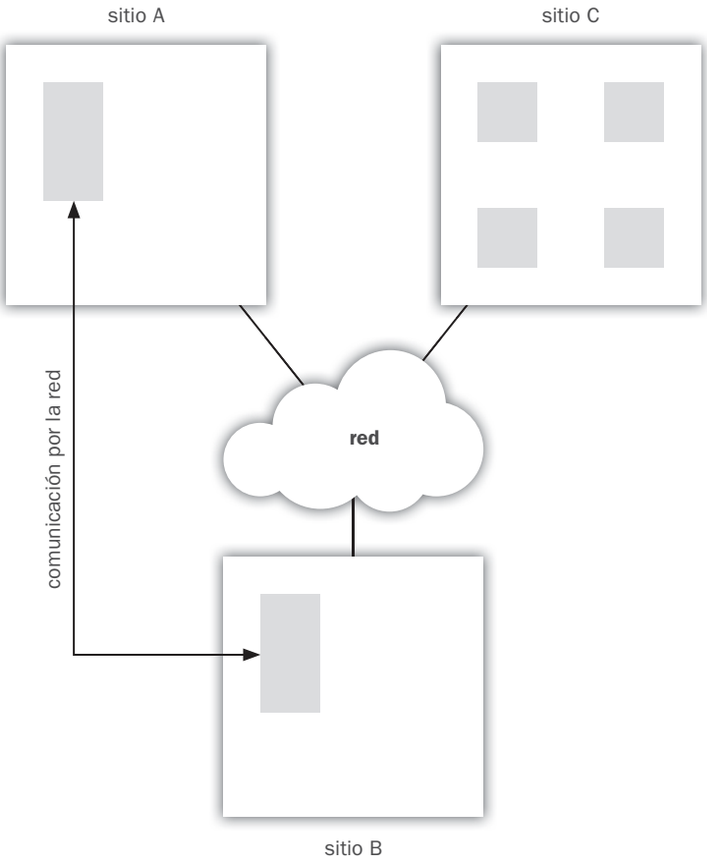


Figura 17.9. Un sistema distribuido.

17.4.2. Aspectos de la implementación

La atomicidad de las transacciones es un aspecto importante de la construcción de un sistema distribuido de bases de datos. Si una transacción se ejecuta usando dos sitios, a menos que los diseñadores del sistema sean cuidadosos, puede comprometerse en un sitio y cancelarse en otro, lo que conduciría a un estado de inconsistencia. Los protocolos de compromiso de transacciones aseguran que tales situaciones no se produzcan. El *protocolo de compromiso de dos* fases (C2F) es el más utilizado de estos protocolos.

La idea básica de C2F es que cada sitio ejecuta la transacción justo hasta que entra en el estado parcialmente comprometida, y entonces deja la decisión del compromiso a un único sitio coordinador; se dice que en ese punto la transacción está en estado *preparada* en el sitio. El coordinador solo decide comprometer la transacción si la transacción alcanza el estado preparada en todos los sitios en los que se ejecutó; en otro caso (por ejemplo, si la transacción se canceló en algún sitio), el coordinador decide cancelar la transacción. Todos los sitios en los que la transacción se ejecutó deben acatar la decisión del coordinador. Si un sitio falla cuando una transacción se encuentra en estado preparada, cuando el sitio se recupere del fallo debería estar en posición de comprometer o cancelar la transacción, dependiendo de la decisión del coordinador. El protocolo C2F se describe en detalle en la Sección 19.4.1.

El control de concurrencia es otra característica de una base de datos distribuida. Como una transacción puede acceder a elementos de datos de varios sitios, los administradores de transacciones de varios sitios pueden necesitar coordinarse para implementar el control de concurrencia. Si se utiliza bloqueo, se puede realizar de forma local en los sitios que contienen los elementos de datos accedidos, pero también existe la posibilidad de un interbloqueo que involucre a transacciones originadas en múltiples sitios. Por lo tanto, es necesario llevar la detección de interbloqueos entre múltiples sitios. Los fallos son más comunes en los sistemas distribuidos, dado que no solo pueden fallar las computadoras, sino que también pueden fallar los enlaces de comunicaciones. La réplica de los elementos de datos, que es la clave para el funcionamiento continuado de las bases de datos distribuidas cuando se producen fallos, complica aún más el control de concurrencia. La Sección 19.5 proporciona más detalles sobre el control de concurrencia en bases de datos distribuidas.

Los modelos estándar de transacciones, basados en múltiples acciones llevadas a cabo por una única unidad de programa, son a menudo inapropiadas para realizar tareas que cruzan los límites de las bases de datos que no pueden o no cooperarán para implementar protocolos como C2F. Para estas tareas se utilizan generalmente técnicas alternativas, basadas en *mensajería persistente* para las comunicaciones; la mensajería persistente se trata en la Sección 19.4.3.

Cuando las tareas a realizar son complejas, involucrando múltiples bases de datos y/o múltiples interacciones con personas, la coordinación de las tareas y el aseguramiento de las propiedades de las transacciones para las tareas se vuelven más complicados. Los *sistemas de gestión de flujos de trabajo* son sistemas diseñados para ayudar en la realización de dichas tareas, y se describen en la Sección 26.2.

En caso de que una empresa tenga que escoger entre una arquitectura distribuida y una arquitectura centralizada para implementar una aplicación, el arquitecto del sistema debe sopesar las ventajas e inconvenientes de la distribución de datos. Ya se han visto las ventajas de utilizar bases de datos distribuidas. El principal inconveniente de los sistemas distribuidos de bases de datos es la complejidad añadida que es necesaria para garantizar la coordinación apropiada entre los sitios. Esta creciente complejidad tiene varias facetas:

- **Coste de desarrollo del software.** La implementación de un sistema distribuido de bases de datos es más difícil y, por tanto, más costoso.
- **Mayor probabilidad de errores.** Como los sitios que constituyen el sistema distribuido funcionan en paralelo, es más difícil asegurar la corrección de los algoritmos del funcionamiento especial durante los fallos de parte del sistema, así como de la recuperación. Son probables errores extremadamente sutiles.
- **Mayor sobrecarga de procesamiento.** El intercambio de mensajes y el cómputo adicional necesario para la coordinación entre los distintos sitios constituyen una forma de sobrecarga que no surge en los sistemas centralizados.

Existen varios enfoques acerca del diseño de las bases de datos distribuidas que abarcan desde los diseños completamente distribuidos hasta los que incluyen un alto grado de centralización. Se estudiarán en el Capítulo 19.

17.5. Tipos de redes

Las bases de datos distribuidas y los sistemas cliente-servidor se construyen en torno a las redes de comunicación. Existen básicamente dos clases de redes: las **redes de área local** y las **redes de área extensa.** La diferencia principal entre ambas es la forma en que están distribuidas geográficamente. Las redes de área local están compuestas por procesadores distribuidos en áreas geográficas pequeñas, tales como un único edificio o varios edificios adyacentes. Por su parte, las redes de área extensa se componen de un número determinado de procesadores autónomos que están distribuidos a lo largo de una extensa área geográfica (como puede ser un país o todo el mundo). Estas diferencias implican importantes variaciones en la velocidad y en la fiabilidad de la red de comunicación y quedan reflejadas en el diseño del sistema operativo distribuido.

17.5.1. Redes de área local

Las **redes de área local** (LAN: *Local Area Network*) (Figura 17.10) surgen a principios de los años setenta como una forma de comunicación y de compartimiento de datos entre varias computadoras. La gente se dio cuenta de que en muchas empresas era más económico tener muchas computadoras pequeñas, cada una de ellas con sus propias aplicaciones, que un enorme y único sistema. La conexión de estas pequeñas computadoras formando una red parece un paso natural porque, probablemente, cada pequeña computadora necesite acceder a un conjunto complementario de dispositivos periféricos (como discos e impresoras) y porque en una empresa suele ser necesario el compartimiento de algunos datos.

Las LAN se utilizan generalmente en un entorno de oficina. Todos los puestos de estos sistemas están próximos entre sí, por lo que los enlaces de comunicación suelen poseer una mayor velocidad y una tasa de errores más baja que la que se da en las redes de área extensa. Los enlaces más comunes en una red de área local son el par trenzado, el cable coaxial, la fibra óptica y, cada vez más, las conexiones inalámbricas. La velocidad de comunicación varía desde decenas de megabits por segundo (en las redes de área local inalámbricas), hasta un gigabit por segundo para Gigabit Ethernet. La norma más reciente de Ethernet es Ethernet de 10 gigabits.

Una **red de área de almacenamiento** (SAN: *Storage Area Network*) es un tipo especial de red de área local de alta velocidad destinada a conectar numerosos bancos de dispositivos de almacenamiento (discos) a las computadoras que utilizan los datos (véase la Figura 17.11).

Por tanto, las redes de área de almacenamiento ayudan a construir *sistemas de discos compartidos* a gran escala. El motivo de utilizar redes de área de almacenamiento para conectar múltiples computadoras a grandes bancos de dispositivos de almacenamiento es esencialmente el mismo que para las bases de datos de disco compartido:

- Ampliabilidad, añadiendo más computadoras.
- Alta disponibilidad, ya que los datos son accesibles incluso aunque falle alguna computadora.

Las organizaciones RAID se utilizan en dispositivos de almacenamiento para asegurar la alta disponibilidad de los datos, permitiendo continuar el procesamiento incluso aunque falle algún disco. Las redes de área de almacenamiento normalmente se construyen con redundancia, como múltiples caminos entre nodos; así, si un componente como un enlace o una conexión a la red fallan, la red continúa funcionando.

17.5.2. Redes de área extensa

Las **redes de área extensa** (WAN: *Wide Area Networks*) surgen a finales de los sesenta principalmente como un proyecto de investigación académica para proporcionar una comunicación eficiente entre varios lugares, permitiendo que una gran comunidad de usuarios pudiera compartir hardware y software de una manera conveniente y económica. A principios de los años sesenta se desarrollaron sistemas que permitían que terminales remotos se conectaran a una computadora central a través de la línea telefónica, pero no eran verdaderas WAN. *Arpanet* fue la primera WAN que se diseñó y se desarrolló. El trabajo en Arpanet comenzó en 1968. Arpanet ha crecido de tal forma que ha pasado de ser una red experimental de cuatro puestos a una red de redes extendida por todo el mundo, **Internet,** abarcando a cientos de millones de sistemas de computación. Los enlaces típicos de Internet son líneas de fibra óptica y, a veces, canales vía satélite. Las transferencias de datos para los enlaces de área extensa varían normalmente entre los pocos megabits por segundo y los cientos de gigabits por segundo. El último enlace, hasta el puesto del usuario, solía ser el más lento; se basa a menudo en la tecnología de *línea de suscriptor digital* (DSL: *Digital Subscriber Line*) (de unos pocos megabits por segundo), o módem de cable (de hasta 10 megabits por segundo) o conexiones de módem telefónico sobre líneas telefónicas (de hasta 56 kilobits por segundo), o una conexión inalámbrica de varios megabits por segundo.

Además de los límites en las velocidades de transmisión, las comunicaciones en una WAN también tienen que tratar con la **latencia:** un mensaje puede tardar unos cientos de milisegundos en transmitirse y entregarse en una red que cruce todo el mundo, tanto debido a los retrasos por la velocidad de la luz, como debido a los retardos en las colas de los encaminadores (*router*[3]) en el trayecto del mensaje. Las aplicaciones cuyos datos y recursos de computación se encuentran distribuidos geográficamente tienen que diseñarse cuidadosamente para asegurar que la latencia no influye excesivamente en el rendimiento del sistema.

Las WAN se pueden clasificar en dos tipos:

- En las WAN de **conexión discontinua,** como las basadas en conexiones por radio, las computadoras solo están conectadas a la red durante intervalos de tiempo.
- En las WAN de **conexión continua,** como Internet por cable, las computadoras están conectadas a la red continuamente.

Las redes que no están continuamente conectadas no suelen permitir las transacciones entre distintos sitios, pero pueden almacenar copias locales de los datos remotos y actualizarlas periódicamente (por ejemplo, todas las noches). Para las aplicaciones en las que la consistencia no es un factor crítico, como ocurre en el compartimiento de documentos, los sistemas de software de grupo como Lotus Notes permiten realizar localmente las actualizaciones de los datos remotos y propagar más tarde dichas actualizaciones al sitio remoto. Debe detectarse y resolverse el riesgo potencial de conflicto entre varias actualizaciones realizadas en sitios diferentes. Más adelante, en la Sección 25.5.4, se describe un mecanismo para detectar actualizaciones conflictivas; el mecanismo de resolución de actualizaciones conflictivas es, sin embargo, dependiente de la aplicación.

3 *Router* se ha introducido como anglicismo en el lenguaje coloquial del usuario. También se conoce como «rúter», «enrutador», «ruteador» o «encaminador», traducción adoptada en este texto. (*N. del E.*)

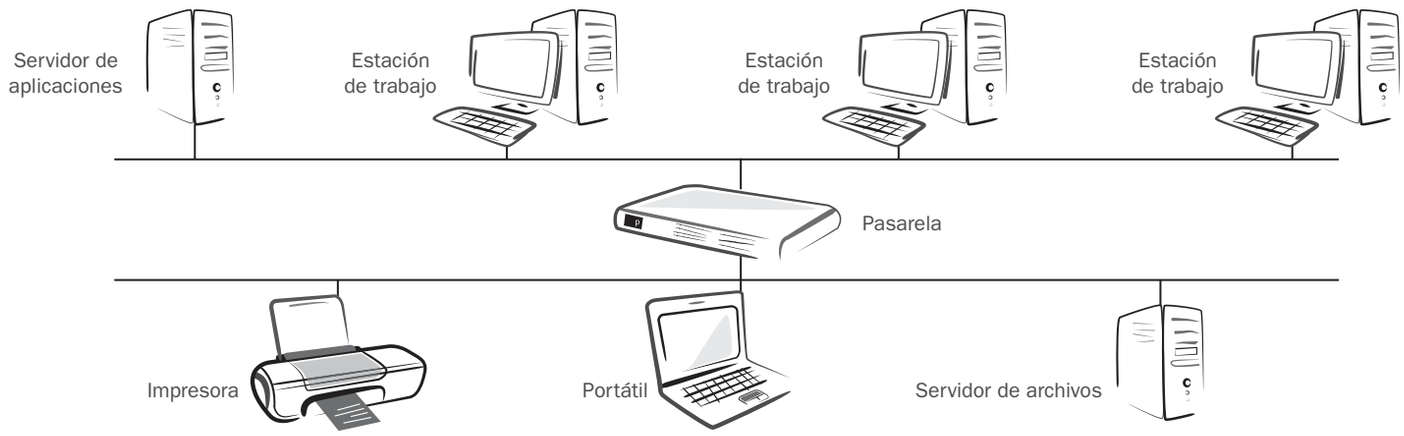


Figura 17.10. Red de área local.

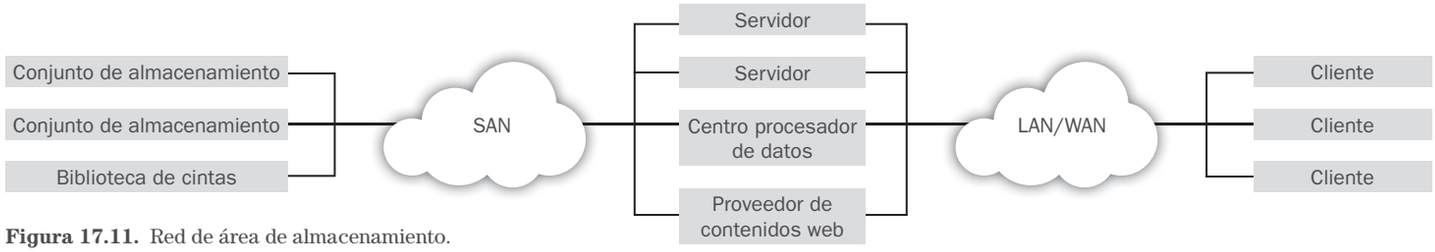


Figura 17.11. Red de área de almacenamiento.

17.6. Resumen

- Los sistemas de bases de datos centralizados se ejecutan completamente en una única computadora. Con el crecimiento de las computadoras personales y las redes de área local, se ha ido desplazando hacia el lado del cliente la funcionalidad de la fachada de la base de datos, de modo que los sistemas servidores provean la funcionalidad del sistema subyacente. Los protocolos de interfaz cliente-servidor han ayudado al crecimiento de los sistemas de bases de datos cliente-servidor.
- Los servidores pueden ser servidores de transacciones o servidores de datos, aunque el uso de los servidores de transacciones excede ampliamente el uso de los servidores de datos para proporcionar servicios de bases de datos.
  - Los servidores de transacciones tienen múltiples procesos, ejecutándose posiblemente en múltiples procesadores. Dado que estos procesos tienen acceso a los datos comunes, como la memoria intermedia de la base de datos, los sistemas almacenan dichos datos en memoria compartida. Además de los procesos que gestionan consultas, hay procesos del sistema que realizan tareas como la gestión de los bloqueos y del registro histórico y los puntos de revisión.
  - Los sistemas servidores de datos suministran datos sin formato a los clientes. Tales sistemas se esfuerzan en minimizar la comunicación entre clientes y servidores usando una caché de datos y de bloqueos en los clientes. Los sistemas paralelos de bases de datos utilizan optimizaciones similares.
- Los sistemas de bases de datos paralelos constan de varios procesadores y varios discos conectados a través de una red de interconexión de alta velocidad. La ganancia de velocidad mide cuánto puede incrementarse la velocidad de procesamiento al incrementarse el paralelismo dada una transacción. La ampliabilidad mide lo bien que se gestiona un mayor número de transacciones cuando se incrementa el paralelismo. La interferencia, el sesgo y los costes de inicio actúan como barreras para obtener la ganancia de velocidad y la ampliabilidad ideales.
- Las arquitecturas de bases de datos paralelas pueden clasificarse en arquitecturas de memoria compartida, de disco compartido, sin compartimiento o jerárquicas. Estas arquitecturas tienen distintos compromisos entre la ampliabilidad y la velocidad de comunicación.
- Un sistema de bases de datos distribuido es un conjunto de bases de datos parcialmente independientes que (idealmente) comparten un esquema común y coordinan el procesamiento de transacciones que acceden a datos remotos. Los sistemas se comunican entre sí a través de una red de comunicación.
- Las redes de área local conectan nodos que están distribuidos sobre áreas geográficas pequeñas, tales como un único edificio o varios edificios adyacentes. Las redes de área extensa conectan nodos a lo largo de una extensa área geográfica. Actualmente, la red de área amplia más extensa que se utiliza es Internet.
- Las redes de área de almacenamiento son un tipo especial de redes de área local diseñadas para proporcionar interconexión rápida entre grandes bancos de dispositivos de almacenamiento y múltiples computadoras.

Términos de repaso

- Sistemas centralizados.
- Sistemas servidores.
- Paralelismo de grano grueso.
- Paralelismo de grano fino.
- Estructura de proceso de la base de datos.
- Exclusión mutua.
- Hilo (hebra).
- Procesos del servidor.
  - Proceso administrador de bloqueos.
  - Proceso escritor de bases de datos.
  - Proceso escritor de registros.
  - Proceso punto de revisión.
  - Proceso monitor de procesos.
- Sistemas cliente-servidor.
- Servidor de transacciones.
- Servidor de consultas.
- Servidor de datos.
  - Preextracción.
  - Liberación de bloqueos.
  - Caché de datos.
  - Coherencia de caché.
  - Caché de bloqueos.
  - Retrollamada.
- Sistemas paralelos.
- Rendimiento.
- Tiempo de respuesta.
- Ganancia de velocidad.
  - Lineal.
  - Sublineal.
- Ampliabilidad.
  - Lineal.
  - Sublineal.
  - Por lotes.
  - De transacciones.
- Costes de inicio.
- Interferencia.
- Sesgo.
- Redes de interconexión.
  - Bus.
  - Malla.
  - Hipercubo.
- Arquitecturas paralelas de bases de datos.
  - Memoria compartida.
  - Disco compartido (agrupaciones).
  - Sin compartimiento.
  - Jerárquica.
- Tolerancia ante fallos.
- Memoria virtual distribuida.
- Arquitectura de memoria no uniforme (NUMA).
- Sistemas distribuidos.
- Bases de datos distribuidas.
  - Sitios (nodos).
  - Transacción local.
  - Transacción global.
  - Autonomía local.
- Sistema con múltiples bases de datos.
- Tipos de redes.
  - Redes de área local (LAN).
  - Redes de área extensa (WAN).
  - Redes de área de almacenamiento (SAN).

Ejercicios prácticos

17.1. En lugar de almacenar estructuras compartidas en memoria compartida, una arquitectura alternativa podría ser almacenarlas en la memoria local de un proceso especial, y acceder a los datos compartidos mediante comunicación interprocesal con ese proceso. ¿Cuál sería la desventaja de dicha arquitectura?

17.2. La máquina que hace de servidor en los sistemas cliente-servidor típicos es mucho más potente que los clientes, es decir, su procesador es más rápido, puede tener varios procesadores, tiene más memoria y tiene discos de mayor capacidad. En vez de este escenario, considere el caso en el que los clientes y el servidor tuvieran exactamente la misma potencia. ¿Tendría sentido construir un sistema cliente-servidor en ese caso? ¿Por qué? ¿Qué caso se ajustaría mejor a una arquitectura servidora de datos?

17.3. Considere un sistema de base de datos orientada a objetos sobre una arquitectura cliente-servidor en la que el servidor actúa como servidor de datos.

a. ¿Cuál es el efecto de la velocidad de interconexión entre el cliente y el servidor en los casos de envío de páginas y de objetos?

b. Si se utiliza envío de páginas, la caché de datos en el cliente puede organizarse como una caché de tuplas o una caché de páginas. La caché de páginas almacena los datos en unidades de páginas, mientras que la caché de tuplas almacena los datos en unidades de tuplas. Suponga que las tuplas son más pequeñas que una página. Describa una ventaja de la caché de tuplas frente a la caché de páginas.

17.4. Suponga una transacción escrita en C con código SQL incorporado que ocupa el 80 % del tiempo en ejecutar el código SQL y solo el 20 % restante en el código C. ¿Qué ganancia de velocidad puede esperarse si solo se hace paralelo el código SQL?

Justifique la respuesta.

17.5. Algunas operaciones de bases de datos, como la reunión, se pueden ver con una diferencia significativa de velocidad cuando los datos (por ejemplo, una de las relaciones involucradas en la reunión) coinciden en memoria, comparada con la situación en la que los datos no coinciden en memoria. Demuestre cómo este hecho puede explicar el fenómeno de **ganancia de velocidad superlineal,** en el que una aplicación observa una ganancia en velocidad mayor que la cantidad de recursos que se le han asignado.

17.6. Los sistemas paralelos suelen tener una estructura de red en la que un conjunto de n procesadores se conectan a un mismo conmutador de Ethernet. Esta arquitectura ¿se corresponde con un bus, una malla o un hipercubo? Si no, ¿cómo describiría esta arquitectura de interconexión?

Ejercicios

17.7. ¿Por qué es relativamente fácil trasladar una base de datos desde una máquina con un único procesador a otra con varios procesadores si no es necesario hacer paralelas las consultas individuales?

17.8. Las arquitecturas servidoras de transacciones son populares entre las bases de datos relacionales cliente-servidor, en las que las transacciones son cortas. Por otra parte, las arquitecturas servidoras de datos son populares entre los sistemas cliente-servidor de bases de datos orientadas a objetos, en las que las transacciones son relativamente largas. Indique dos razones por las que los servidores de datos puedan ser populares entre las bases de datos orientadas a objetos y no lo sean entre las bases de datos relacionales.

17.9. ¿Qué es la liberación de bloqueos y bajo qué condiciones es necesaria? ¿Por qué no es necesario si la unidad de envío de datos es un elemento?

17.10. Suponga que se encuentra a cargo de las operaciones de la base de datos de una empresa cuyo trabajo principal es el de procesar transacciones. Suponga además que la empresa crece rápidamente cada año y que el sistema informático actual se ha quedado pequeño. Cuando se escoja una nueva computadora paralela, ¿qué factor será más importante: la ganancia de velocidad, la ampliabilidad por lotes o la ampliabilidad de transacciones? ¿Por qué?

17.11. Los sistemas de bases de datos normalmente se implementan como un conjunto de procesos (o hilos) que comparten un área de memoria.

a. ¿Cómo se controla el acceso a la memoria compartida?

b. ¿Es apropiado el bloqueo de dos fases para secuenciar el acceso a las estructuras de datos en la memoria compartida? Explique la respuesta.

17.12. ¿Es apropiado permitir que un proceso de usuario acceda al área de memoria compartida de un sistema de base de datos? Justifique su respuesta.

17.13. En un sistema de procesamiento de transacciones, ¿cuáles son los factores que afectan negativamente la ampliabilidad lineal? ¿Cuál de esos factores es probablemente el más importante en cada una de estas arquitecturas: memoria compartida, disco compartido y sin compartimiento?

17.14. Los sistemas de memoria se pueden dividir en varios módulos, cada uno de los cuales puede servir una consulta separada en un momento dado. ¿Qué impacto tendría esta arquitectura de memoria en el número de procesadores que se pueden disponer en un sistema de memoria compartida?

17.15. Considere un banco que dispone de un conjunto de sitios en los que se ejecuta un sistema de base de datos. Suponga que la transferencia electrónica de dinero entre ellos es el único modo de interacción entre las bases de datos. ¿Puede tal sistema ser calificado como distribuido? ¿Por qué?

Notas bibliográficas

Hennessy et ál. [2002] proporcionan una excelente introducción al área de la arquitectura de computadoras. Abadi [2009] proporciona una excelente introducción a la computación en la nube y los retos que plantea la ejecución de transacciones de bases de datos en este entorno.

Gray y Reuter [1993] ofrecen en su libro una descripción del procesamiento de transacciones, incluyendo la arquitectura cliente-servidor y los sistemas distribuidos. Las notas bibliográficas del Capítulo 5 proporcionan referencias para más información sobre ODBC, JDBC y otras interfaces de programación para aplicaciones.

DeWitt y Gray [1992] describen los sistemas paralelos de bases de datos, incluyendo sus propias arquitecturas y medidas de rendimiento. Duncan [1990] introduce las arquitecturas paralelas de computadoras. Dubois y Thakkar [1992] elaboran una colección de artículos sobre las arquitecturas ampliables de memoria compartida. Las agrupaciones DEC con Rdb constituyen uno de los primeros usuarios comerciales de la arquitectura de bases de datos de disco compartido. Rdb es ahora propiedad de Oracle y se denomina Oracle Rdb. La máquina de base de datos Teradata fue uno de los primeros sistemas comerciales que utilizaron la arquitectura sin compartimiento de bases de datos. También se construyeron sobre arquitecturas sin compartimiento los prototipos de investigación Grace y Gamma.

El libro de texto de Ozsu y Valduriez [1999] trata los sistemas distribuidos de bases de datos. Pueden encontrarse más referencias sobre los sistemas de bases de datos paralelos y distribuidos en las notas bibliográficas de los Capítulos 18 y 19, respectivamente.

Comer, Halsall [2006] y Thomas [1996] describen las redes de computadoras e Internet. Tanenbaum [2002], Kurose y Ross [2005], y Peterson y Davie [2007] proporcionan revisiones generales de las redes de computadoras.

Bases de datos paralelas 18

En este capítulo se estudian los algoritmos fundamentales de los sistemas de bases de datos paralelas basados en el modelo de datos relacional. En concreto, este capítulo se centra en la ubicación de los datos en varios discos y en la evaluación en paralelo de las operaciones relacionales, dos conceptos que han sido esenciales para el éxito de las bases de datos paralelas.

18.1. Introducción

En algún momento de hace unas dos décadas los sistemas paralelos de bases de datos habían sido casi descartados, incluso por algunos de sus más firmes partidarios. Actualmente los venden con éxito casi todas las marcas de bases de datos. Este cambio ha sido impulsado por ciertas tendencias:

- Los requisitos transaccionales de las empresas han aumentado con el empleo creciente de las computadoras. Además, el incremento del World Wide Web ha creado muchos sitios con millones de visitantes, y la creciente cantidad de datos obtenidos de esos visitantes ha generado en muchas empresas bases de datos enormes.
- Las empresas utilizan volúmenes crecientes de datos —como los relativos a lo que se compra, los enlaces web que se pulsan o la hora a la que se realizan las llamadas telefónicas— para planificar sus actividades y sus tarifas. Las consultas utilizadas para estos fines se denominan **consultas de ayuda a la toma de decisiones** y pueden llegar a necesitar varios terabytes de datos. Los sistemas con un solo procesador no son capaces de tratar tales volúmenes de datos a la velocidad necesaria.
- La naturaleza orientada a conjuntos de las consultas a las bases de datos se presta de modo natural a la paralelización. Diferentes sistemas comerciales y experimentales han demostrado la potencia y la capacidad de crecimiento del procesamiento paralelo de las consultas.
- Al abaratarse los microprocesadores, las máquinas paralelas se han popularizado y se han vuelto relativamente más baratas.
- Los propios procesadores individuales se han convertido en máquinas paralelas que utilizan arquitecturas multinúcleo.

Como se ha estudiado en el Capítulo 17, el paralelismo se utiliza para mejorar la velocidad, pues las consultas se ejecutan más rápido al disponer de más recursos, como procesadores y discos. El paralelismo también se utiliza para proporcionar capacidad de crecimiento, pues la creciente carga de trabajo se consigue tratar sin incrementar el tiempo de respuesta, con un aumento del grado de paralelismo.

En el Capítulo 17 se esbozaron las diferentes arquitecturas de los sistemas paralelos de bases de datos: de memoria compartida, de discos compartidos, sin compartimiento y de arquitecturas jerárquicas. En resumen, en las arquitecturas de memoria compartida todos los procesadores comparten memoria y discos; en las arquitecturas de disco compartido, los procesadores tienen memorias independientes pero comparten los discos; en las arquitecturas sin compartimiento, los procesadores no comparten ni la memoria ni los discos; y las arquitecturas jerárquicas tienen nodos que no comparten entre sí ni la memoria ni los discos, pero cada nodo tiene internamente una arquitectura de memoria o de disco compartidos.

18.2. Paralelismo de E/S

En su forma más sencilla, el término **paralelismo de E/S** se refiere a la división de las relaciones entre varios discos para reducir el tiempo necesario para recuperarlos. La forma más habitual de división de datos en un entorno de bases de datos paralelas es la *división horizontal*. En la **división horizontal,** las tuplas de cada relación se dividen (o desagrupan) entre varios discos, de modo que cada tupla resida en uno distinto. Se han propuesto varias estrategias de división.

18.2.1. Técnicas de división

Se presentan tres estrategias básicas para la división de datos. Suponga que hay n discos, $D_0, D_1, \ldots, D_{n-1}$, entre los cuales se van a dividir los datos.

- **Turno rotatorio.** La relación se explora en un orden cualquiera y la i-ésima tupla se envía al disco numerado $D_{i \bmod n}$. El esquema de turno rotatorio asegura una distribución homogénea de las tuplas entre los discos; es decir, cada disco tiene aproximadamente el mismo número de tuplas que los demás.
- **División por asociación.** En esta estrategia de desagrupación, uno o varios atributos del esquema de la relación dada se designan como atributos de la división. Se escoge una función de asociación cuyo rango sea $\{0, 1, \ldots, n-1\}$. Cada tupla de la relación original se asocia en términos de los atributos de la división. Si la función de asociación devuelve i, la tupla se ubica en el disco D_i.[1]
- **División por rangos.** Esta estrategia distribuye rangos contiguos de valores de los atributos a cada disco. Se escoge un atributo de división, A, como **vector de división.** La relación se divide de la siguiente manera: sea $[v_0, v_1, \ldots, v_{n-2}]$ el vector de división, tal que, si $i < j$, entonces $v_i < v_j$. Considere una tupla t tal que $t[A] = x$. Si $x < v_0$, entonces t se ubica en el disco D_0. Si $x \geq v_{n-2}$, entonces t se ubica en el disco D_{n-1}. Si $v_i \leq x < v_{i+1}$, entonces t se ubica en el disco D_{i+1}.

  Por ejemplo, en una división por rangos con tres discos numerados del cero al dos se pueden asignar las tuplas con valores menores que cinco al disco cero, las de valores entre cinco y cuarenta al disco uno, y las que tienen valores mayores de cuarenta al disco dos.

1 El diseño de funciones de asociación se ha visto en la Sección 11.6.1.

18.2.2. Comparación de las técnicas de división

Una vez dividida una relación entre varios discos, se puede recuperar en paralelo de todos ellos. De modo parecido, cuando se está dividiendo una relación, se puede escribir en paralelo en varios discos. De esta manera, las velocidades de transferencia de lectura o de escritura de la relación completa son mucho mayores con paralelismo de E/S que sin él. Sin embargo, la lectura de la relación completa, o la *exploración de la relación,* es solo uno de los tipos de acceso a los datos. El acceso a los datos puede clasificarse de la siguiente forma:

1. Exploración de la relación completa.
2. Localización de tuplas de manera asociativa (por ejemplo, *nombre_empleado* = «Campbell»); estas consultas, denominadas **consultas concretas,** buscan tuplas que tengan un valor concreto para un atributo determinado.
3. Localización de todas las tuplas cuyo valor de un atributo dado se halle en un rango especificado (por ejemplo, 10.000 < *sueldo* < 20.000); estas consultas se denominan **consultas de rango.**

Las diferentes técnicas de división permiten estos tipos de acceso con diversos niveles de eficacia:

- **Turno rotatorio.** El esquema se adapta perfectamente a las aplicaciones que desean leer secuencialmente la relación completa en cada consulta. Con este esquema, tanto las consultas concretas como las de rango son difíciles de procesar, ya que se deben emplear en la búsqueda todos y cada uno de los n discos.
- **División por asociación.** Este esquema se adapta mejor a las consultas concretas basadas en el atributo de división. Por ejemplo, si se divide una relación en términos del atributo *número_teléfono*, se puede responder a la consulta «Buscar el registro del empleado con *número_teléfono* = 555-3333» aplicando la función de división por asociación a 555-3333 y buscando luego en ese disco. Dirigir la consulta a un solo disco ahorra el coste inicial de comenzar una consulta en varios discos y deja a los demás discos libres para procesar otras consultas.

  La división por asociación también resulta útil para las exploraciones secuenciales de toda la relación. Si la función de asociación es una buena función aleatoria y los atributos de división constituyen una clave de la relación, el número de tuplas en cada uno de los discos es aproximadamente el mismo, sin mucha varianza. Por tanto, el tiempo empleado para explorar la relación es aproximadamente $1/n$ del tiempo necesario para explorar la relación en un sistema de disco único.

  El esquema, sin embargo, no se adapta bien a las búsquedas concretas en términos de atributos que no sean de división. La división basada en asociación tampoco resulta muy adecuada para las respuestas a consultas de rangos, dado que, generalmente, las funciones de asociación no conservan la proximidad dentro de ellos. Por tanto, hace falta explorar todos los discos para responder a las consultas de rango.
- **División por rangos.** Este esquema se adapta bien a las consultas concretas y de rango basadas en el atributo de división. Para las consultas concretas, se puede analizar el vector de división para encontrar el disco en el que reside la tupla. En las consultas de rango se consulta el vector de división para hallar el rango de discos en que pueden residir las tuplas. En ambos casos la búsqueda se limita exactamente a aquellos discos que pueden tener tuplas de interés.

  Una ventaja es que si solo hay unas cuantas tuplas en el rango consultado, la consulta se suele enviar a un único disco, en vez de a todos. Dado que se pueden utilizar otros discos para responder a otras consultas, la división por rangos da lugar a un mayor rendimiento de las consultas al tiempo que se mantiene un buen tiempo de respuesta. Por otro lado, si hay muchas tuplas en el rango consultado (como ocurre cuando el rango consultado es una fracción mayor del dominio de la relación), es necesario recuperar muchas tuplas de pocos discos, lo que origina un cuello de botella de E/S (punto caliente) en esos discos. En este ejemplo de **sesgo de ejecución** todo el procesamiento tiene lugar en una partición (o en solo unas pocas). Por el contrario, la división por asociación y la de turno rotatorio emplearían todos los discos para esas consultas, lo que proporcionaría un tiempo de respuesta menor para aproximadamente el mismo rendimiento.

El tipo de división también afecta a otras operaciones relacionales, como las reuniones, como se verá en la Sección 18.5. De este modo, la elección de la técnica de división también depende de las operaciones que haya que ejecutar. En general, se prefieren las divisiones por asociación y en rangos al turno rotatorio.

En un sistema con muchos discos, el número de discos en los que se divide una relación puede escogerse de la siguiente manera. Si una relación solo contiene unas pocas tuplas que caben en un único bloque de disco, es mejor asignarla a un solo disco. Las relaciones grandes se dividen preferiblemente entre todos los discos disponibles. Si una relación consta de m bloques de disco y hay n discos disponibles en el sistema, se deberán asignar a la relación **mín**(m, n) discos.

18.2.3. Tratamiento de sesgo

La distribución de las tuplas al dividir una relación (excepto mediante el turno rotatorio) puede estar **sesgada,** con un porcentaje alto de tuplas ubicado en algunas divisiones y porcentajes menores en otras. Los diferentes tipos de sesgo se clasifican como:

- Sesgo de los valores de los atributos.
- Sesgo de la división.

El **sesgo de los valores de los atributos** se refiere al hecho de que algunos valores aparezcan en los atributos de división de muchas tuplas. Todas las tuplas con el mismo valor del atributo de división terminan en la misma partición, lo que da lugar al sesgo. El **sesgo de la división** se refiere al hecho de que puede haber un desequilibrio de reparto en la división, aunque no haya sesgo en los atributos.

El sesgo de los valores de los atributos puede dar lugar a una división sesgada independientemente de que se utilice división por rangos o por asociación. Si no se escoge cuidadosamente el vector de división, la división por rangos puede dar lugar a un sesgo de división. El sesgo de división es menos probable en la división por asociación si se ha escogido una buena función de asociación.

Como se indicó en la Sección 17.3.1, incluso un sesgo pequeño puede dar lugar a una disminución significativa del rendimiento. El sesgo es un problema que se agrava al aumentar el grado de paralelismo. Por ejemplo, si una relación de mil tuplas se divide en diez partes y la división está sesgada, puede haber algunas particiones de tamaño menor que cien y otras de tamaño mayor que cien; incluso si una partición llega a tener tamaño doscientos, la aceleración que se obtendría al acceder en paralelo a las particiones solo sería de cinco, en lugar del valor de diez que cabría esperar. Si la misma relación tiene que dividirse en cien partes, cada partición tendrá de media diez tuplas. Si una partición llega a tener cuarenta tuplas (lo que es posible dado el gran número de particiones), el aumento de velocidad que se obtendría al acceder a ellas en paralelo sería de veinticinco, en vez de cien. Por tanto, se puede ver que la pérdida de velocidad debida al sesgo aumenta con el paralelismo.

Se puede construir un **vector de división por rangos equilibrado** mediante ordenación. La relación primero se ordena según los atributos de división. A continuación se explora de forma ordenada. Después de que se haya leído cada $1/n$ de la relación, se añade el valor del atributo de división de la siguiente tupla al vector de división. En este caso, n denota el número de particiones que hay que crear. En caso de que haya muchas tuplas con el mismo

valor para el atributo de división, la técnica puede seguir generando cierto sesgo. El inconveniente principal de este método es la sobrecarga de E/S debida a la ordenación inicial.

La sobrecarga de E/S debida a la construcción de vectores de división por rangos equilibrados se puede reducir creando y almacenando una tabla de frecuencias, o **histograma,** de los valores de los atributos para todos los atributos de cada relación. La Figura 18.1 muestra un ejemplo de histograma para un atributo de tipo entero que toma valores en el rango de uno a veinticinco. Los histogramas ocupan poco espacio, por lo que se pueden almacenar en el catálogo histogramas de varios atributos diferentes. Resulta sencillo crear una función de división por rangos equilibrada dado un histograma de los atributos de división. Si no se almacena el histograma es posible calcularlo de manera aproximada tomando muestras de la relación. Estas muestras son las tuplas de un subconjunto de los bloques de disco elegido aleatoriamente.

Otro enfoque para minimizar el efecto del sesgo, especialmente con la división por rangos, es el empleo de *procesadores virtuales*. En el enfoque de los **procesadores virtuales** se simula que el número de *procesadores virtuales* es múltiplo del número de procesadores reales. Se puede usar cualquiera de las técnicas de división y de evaluación de consultas que se estudiarán posteriormente en este capítulo, asignando las tuplas a los procesadores virtuales en lugar de a los reales. Los procesadores virtuales, a su vez, se hacen corresponder con los procesadores reales, generalmente mediante una división por turno rotatorio.

La idea es que incluso si uno de los rangos tuviera muchas más tuplas que los otros debido al sesgo, estas se podrían repartir entre varios rangos de procesadores virtuales. La asignación por turno rotatorio de los procesadores virtuales a procesadores reales distribuiría el trabajo adicional entre varios procesadores reales, de forma que ningún procesador tuviera que asumir toda la carga.

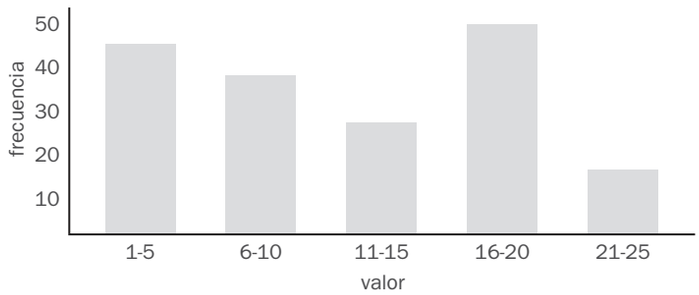


Figura 18.1. Ejemplo de histograma.

18.3. Paralelismo entre consultas

En el **paralelismo entre consultas** se ejecutan en paralelo entre sí diferentes consultas o transacciones. La productividad de las transacciones puede aumentar con esta forma de paralelismo. Sin embargo, el tiempo de respuesta de cada transacción no es menor que si se ejecutara aisladamente. Por ello, la aplicación principal del paralelismo entre consultas es la ampliación de los sistemas de procesamiento de transacciones para permitir un mayor número de transacciones por segundo.

El paralelismo entre consultas es la forma más sencilla de paralelismo que se permite en los sistemas de bases de datos; especialmente en los sistemas paralelos de memoria compartida. Los sistemas de bases de datos diseñados para sistemas con un único procesador pueden utilizarse en arquitecturas paralelas de memoria compartida con pocos cambios o con ninguno, dado que incluso los sistemas secuenciales de bases de datos permiten el procesamiento concurrente. Las transacciones que se habrían realizado de manera concurrente en tiempo compartido en una máquina secuencial se realizan en paralelo en la arquitectura paralela de memoria compartida.

Permitir el paralelismo entre consultas es más complicado en las arquitecturas de disco compartido y sin compartimiento. Los procesadores tienen que realizar algunas tareas, como los bloqueos y el registro histórico, de forma coordinada, y eso exige que se intercambien mensajes. Los sistemas de bases de datos con arquitectura paralela también deben asegurarse de que dos procesadores no actualicen simultáneamente los mismos datos de manera independiente. Además, cuando un procesador accede a los datos o los actualiza, el sistema de bases de datos debe garantizar que tenga su última versión en la memoria intermedia. El problema de asegurar que la versión sea la última disponible se denomina problema de **coherencia de caché.**

Existen varios protocolos para garantizar la coherencia de caché; a menudo los protocolos de coherencia de la caché se integran con los de control de concurrencia de modo que se reduzca la sobrecarga. Uno de los protocolos de este tipo para sistemas de disco compartido actúa de la siguiente forma:

1. Antes de cualquier acceso de lectura o de escritura a una página, la transacción la bloquea en modo compartido o exclusivo, según corresponda. Inmediatamente después de obtener el bloqueo compartido o exclusivo de la página, la transacción lee también su copia más reciente del disco compartido.
2. Antes de que una transacción libere el bloqueo exclusivo de una página, la traslada al disco compartido; posteriormente libera el bloqueo.

Este protocolo garantiza que, cuando una transacción establece un bloqueo compartido o exclusivo sobre una página, obtenga la copia correcta de la página.

Otros protocolos más complejos evitan la lectura y escritura reiteradas del disco exigidas por el protocolo anterior. Estos protocolos no escriben las páginas en el disco cuando se liberan los bloqueos exclusivos. Cuando se obtiene un bloqueo compartido o exclusivo, si la versión más reciente de la página se halla en la memoria intermedia de algún procesador, se obtiene de allí. Estos protocolos se diseñan para tratar peticiones concurrentes. Los protocolos de disco compartido pueden extenderse a las arquitecturas sin compartimiento mediante el siguiente esquema: cada página tiene un **procesador local** P_i y se almacena en el disco D_i. Cuando otros procesadores desean leer la página o escribir en ella, envían las peticiones a su procesador local P_i, dado que no pueden comunicarse directamente con el disco. Las otras acciones son iguales que en los protocolos de disco compartido.

Los sistemas Oracle y Rdb de Oracle son ejemplos de sistemas paralelos de bases de datos de disco compartido que permiten el paralelismo entre consultas.

18.4. Paralelismo en consultas

El **paralelismo en consultas** se refiere a la ejecución en paralelo de una única consulta en varios procesadores y discos. El empleo del paralelismo en consultas es importante para acelerar las consultas de ejecución prolongada. El paralelismo entre consultas no ayuda en esta tarea, dado que cada consulta se ejecuta de manera secuencial.

Para ilustrar la evaluación en paralelo de una consulta, considérese una que exija que se ordene una relación. Suponga que la relación se ha dividido entre varios discos mediante la división por rangos basada en algún atributo y que se solicita la ordenación basada en el atributo de división. La operación de ordenación se puede implementar ordenando cada partición en paralelo y luego se concatenan las particiones ordenadas para obtener la relación ordenada final.

Por tanto, las consultas se pueden hacer paralelas haciendo paralelas las operaciones que las forman. Hay otra fuente de paralelismo para la evaluación de las consultas: el *árbol de operadores* de la consulta puede contener varias operaciones. La evaluación del árbol de operadores se puede hacer paralela evaluando en paralelo las operaciones que no tengan ninguna dependencia entre sí. Además, como se mencionó en el Capítulo 12, puede que se logre encauzar el resultado de una operación hacia otra. Las dos operaciones pueden ejecutarse en paralelo en procesadores separados, uno que genera el resultado que consume el otro, incluso según se genera.

En resumen, hay dos maneras de ejecutar en paralelo una sola consulta:

- **Paralelismo en operaciones.** Se puede acelerar el procesamiento de la consulta haciendo paralela la ejecución de cada una de sus operaciones individuales: ordenación, selección, proyección y reunión. El paralelismo en operaciones se considera en la Sección 18.5.
- **Paralelismo entre operaciones.** Se puede acelerar el procesamiento de la consulta ejecutando en paralelo las diferentes operaciones de las expresiones de las consultas. Esta forma de paralelismo se considera en la Sección 18.6.

Las dos formas de paralelismo son complementarias y pueden utilizarse simultáneamente en una misma consulta. Dado que el número de operaciones de una consulta típica es pequeño comparado con el número de tuplas procesado por cada operación, la primera modalidad puede adaptarse mejor a un aumento del paralelismo. Sin embargo, con el número relativamente pequeño de procesadores de los sistemas paralelos típicos de hoy en día, ambas formas de paralelismo son importantes.

En la siguiente discusión sobre la paralelización de las consultas se da por supuesto que estas son **solo de lectura.** La elección de los algoritmos para la evaluación de las consultas en paralelo depende de la arquitectura de la máquina. En lugar de presentar por separado los algoritmos para cada arquitectura, se utilizará en la descripción un modelo de arquitectura sin compartimiento. Por tanto, se describirá explícitamente el momento en que se deben transferir los datos de un procesador a otro. Este modelo se puede simular con facilidad utilizando las otras arquitecturas, dado que la transferencia de los datos puede realizarse mediante la memoria compartida en las arquitecturas de memoria compartida y mediante los discos compartidos en las arquitecturas de discos compartidos. Por tanto, los algoritmos para las arquitecturas sin compartimiento también pueden utilizarse en las demás arquitecturas. Ocasionalmente se menciona la manera en que se pueden optimizar aún más los algoritmos para los sistemas de memoria o de discos compartidos.

Para simplificar la presentación de los algoritmos se supone que existen n procesadores, $P_0, P_1, \ldots, P_{n-1}$ y n discos, $D_0, D_1 \ldots, D_{n-1}$, donde el disco D_i está asociado con el procesador P_i. Los sistemas reales pueden tener varios discos por cada procesador. No es difícil extender los algoritmos para que permitan varios discos por procesador: basta con permitir que D_i sea un conjunto de discos. Sin embargo, con el objetivo de fomentar la sencillez de la presentación, se supondrá que D_i es un solo disco.

18.5. Paralelismo en operaciones

Dado que las operaciones relacionales trabajan con relaciones que contienen grandes conjuntos de tuplas, las operaciones se pueden paralelizar ejecutándolas sobre subconjuntos diferentes de las relaciones en paralelo. Dado que el número de tuplas de cada relación puede ser grande, el grado de paralelismo es potencialmente enorme. Por tanto, el paralelismo en operaciones resulta natural en los sistemas de bases de datos. En las Secciones 18.5.1 a 18.5.3 se estudian las versiones paralelas de algunas operaciones relacionales frecuentes.

18.5.1. Ordenación paralela

Suponga que se desea ordenar una relación que reside en n discos, $D_0, D_1, \ldots, D_{n-1}$. Si la relación se ha dividido por rangos basándose en los atributos por los que se va a ordenar, entonces, como se indicó en la Sección 18.2.2, se puede ordenar cada partición por separado y concatenar los resultados para obtener la relación completa ordenada. Dado que las tuplas se hallan divididas en n discos, el tiempo necesario para leer la relación completa se reduce gracias al acceso en paralelo.

Si la relación se ha dividido siguiendo algún otro método, se puede ordenar de una de estas dos maneras:

1. Se puede dividir en rangos de acuerdo con los atributos de ordenación y luego ordenar cada partición por separado.
2. Se puede utilizar una versión paralela del algoritmo externo de ordenación-mezcla.

18.5.1.1. Ordenación con división por rangos

La **ordenación con división por rangos** tiene dos etapas: primero, se divide por rangos la relación y, después, se ordena cada una de las particiones. Cuando la relación se ordena mediante división por rangos, no hace falta realizar la división en los mismos procesadores o discos en los que se almacena la relación. Suponga que se escogen los procesadores $P_0, P_1, \ldots, P_m$, donde $m < n$, para ordenar la relación. Esta operación se divide en dos fases:

1. Redistribuir las tuplas de la relación utilizando una estrategia de división por rangos, de manera que todas las tuplas que se hallen dentro del rango i-ésimo se envíen al procesador P_i, que almacena temporalmente la relación en el disco D_i.

   Para implementar la división por rangos cada procesador lee en paralelo las tuplas de su disco y las envía al procesador de destino. Cada procesador $P_0, P_1, \ldots, P_m$ también recibe las tuplas correspondientes a su partición y las almacena localmente. Esta fase necesita una sobrecarga en la E/S de disco y en las comunicaciones.
2. Cada uno de los procesadores ordena localmente su partición de la relación sin interactuar con los demás procesadores. Cada procesador ejecuta la misma operación —por ejemplo, ordenar— sobre un conjunto de datos diferente. (La ejecución de la misma acción en paralelo sobre conjuntos diferentes de datos se denomina **paralelismo de datos**).

   La operación final de mezcla es trivial, ya que la división por rangos de la primera etapa asegura que, para $1 \leq i < j \leq m$, los valores de la clave del procesador P_i son todos menores que los de P_j.

La división por rangos se debe llevar a cabo empleando un buen vector de división por rangos, de manera que cada partición tenga aproximadamente el mismo número de tuplas. Los procesadores virtuales también se pueden utilizar para reducir el sesgo.

18.5.1.2. Ordenación y mezcla externas paralelas

La **ordenación y** la **mezcla externas paralelas** son una alternativa a la división por rangos. Suponga que la relación ya se ha dividido entre los discos $D_0, D_1, \ldots, D_{n-1}$ (no importa la manera en que se haya dividido la relación). La ordenación y mezcla externas paralelas funcionan de la siguiente manera:

1. Cada procesador P_i ordena localmente los datos del disco D_i.
2. El sistema mezcla las partes ordenadas por cada procesador para obtener el resultado ordenado final.

La mezcla de las partes ordenadas del paso dos se puede paralelizar mediante esta secuencia de acciones:

1. El sistema divide en rangos las particiones ordenadas en cada procesador P_i (utilizando el mismo vector de división) entre los procesadores $P_0, P_1, \ldots, P_{m-1}$. Envía las tuplas de acuerdo con el orden establecido, por lo que cada procesador recibe las tuplas en un flujo ordenado.
2. Cada procesador P_i realiza una mezcla de los flujos de datos según los recibe para obtener una sola parte ordenada.
3. Las partes ordenadas de los procesadores $P_0, P_1, \ldots, P_{m-1}$ se concatenan para obtener el resultado final.

Como ya se ha descrito, esta secuencia de acciones da lugar a una variedad interesante del **sesgo de ejecución,** ya que, en primer lugar, cada procesador envía todos los bloques de la división 0 a P_0, después cada procesador envía todos los bloques de la partición 1 a P_1, etc. Así, aunque el envío se produce en paralelo, la recepción de las tuplas es secuencial: primero solo P_0 recibe tuplas, luego solo lo hace P_1, y así sucesivamente. Para evitar este problema, cada procesador envía repetidamente un bloque de datos a cada partición. En otras palabras, cada procesador envía el primer bloque de cada partición, luego envía el segundo bloque de cada partición, etc. En consecuencia, todos los procesadores reciben los datos en paralelo.

Algunas máquinas, como las de la Teradata Purpose-Built Platform Family, utilizan hardware especializado para realizar las mezclas. La red de interconexión BYNET de las máquinas Teradata puede mezclar los resultados de varios procesadores para ofrecer un único resultado ordenado.

18.5.2. Reunión paralela

La operación reunión exige que el sistema compare pares de tuplas para ver si satisfacen la condición de reunión; si la cumplen, añade el par al resultado de la reunión. Los algoritmos de reunión paralela intentan repartir entre varios procesadores los pares que hay que comparar. Cada procesador procesa localmente parte de la reunión. Después, el sistema reúne los resultados de cada procesador para generar el resultado final.

18.5.2.1. Reunión por división

Para ciertos tipos de reuniones, como las equirreuniones y las reuniones naturales, es posible *dividir* las dos relaciones de entrada entre los procesadores y procesar localmente la reunión en cada uno de ellos. Suponga que se utilizan n procesadores y que las relaciones que hay que reunir son r y s. La **reunión por división** funciona de esta forma: el sistema divide las relaciones r y s en n particiones, denominadas $r_0, r_1, \ldots, r_{n-1}$ y $s_0, s_1, \ldots, s_{n-1}$; y envía las particiones r_i y s_i al procesador P_i, donde la reunión se procesa localmente.

La técnica anterior solo funciona correctamente si la reunión es una equirreunión (por ejemplo, $r \bowtie_{r.A=s.B} s$) y se dividen r y s utilizando la misma función de división para sus atributos de reunión. La idea de la división es exactamente la misma que subyace en la fase de división de la reunión por asociación. Sin embargo, en la reunión por división hay dos maneras diferentes de dividir r y s:

- División por rangos de los atributos de reunión.
- División por asociación de los atributos de reunión.

En ambos casos se debe utilizar la misma función de división para las dos relaciones. Para la división por rangos, el mismo vector de división. En el caso de la división por asociación, la misma función de asociación. La Figura 18.2 muestra la división utilizada en una reunión por división paralela.

Una vez divididas las relaciones, se puede utilizar localmente cualquier técnica de reunión en cada procesador P_i para calcular la reunión de r_i y s_i. Por ejemplo, se puede emplear la reunión por asociación, por mezcla o con bucles anidados. Por tanto, se puede utilizar la división para paralelizar cualquier técnica de reunión.

Si alguna de las relaciones r y s, o las dos, ya están divididas basándose en los atributos de reunión (mediante división por asociación o por rangos), el trabajo necesario para la división se reduce mucho. Si las relaciones no están divididas, o lo están de acuerdo con atributos distintos de los de la reunión, hay que volver a dividir las tuplas. Cada procesador P_i lee las tuplas del disco D_i, procesa para cada tupla t la partición j a la que pertenece i y la envía al procesador P_j. El procesador P_j almacena las tuplas en el disco D_j.

Se puede optimizar el algoritmo de reunión utilizado localmente en cada procesador para reducir la E/S guardando en la memoria intermedia algunas de las tuplas, en lugar de escribirlas en el disco. Estas optimizaciones se describen en la Sección 18.5.2.3.

El sesgo representa un problema especial cuando se utiliza la división por rangos, dado que un vector de división que divida una relación de la reunión en particiones de igual tamaño puede dividir las demás relaciones en particiones de tamaño muy variable. El vector de división debe ser tal que $|r_i| + |s_i|$ (es decir, la suma de los tamaños de r_i y s_i) sea aproximadamente igual para todo $i = 0, 1, \ldots, n-1$. Con una buena función de asociación, la división por asociación probablemente tenga menos sesgo, excepto cuando haya muchas tuplas con los mismos valores de los atributos de reunión.

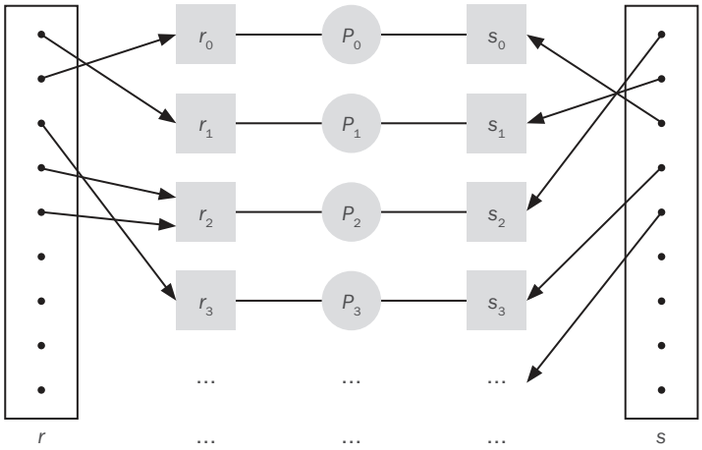


Figura 18.2. Reunión por división paralela.

18.5.2.2. Reunión con fragmentos y réplicas

La división no es aplicable a todos los tipos de reuniones. Por ejemplo, si la condición de reunión es una desigualdad, como $r \bowtie_{r.a<s.b} s$, es posible que todas las tuplas de r se reúnan con alguna tupla de s, y viceversa. Por tanto, puede que no haya un medio sencillo de dividir r y s de modo que las tuplas de la partición r_i solo se reúnan con tuplas de la partición s_i.

Esas reuniones se pueden paralelizar utilizando una técnica denominada *fragmentos y réplicas*. En primer lugar se considerará un caso especial de fragmentos y réplicas —la **reunión con fragmentos y réplicas asimétricos**— que funciona de la siguiente forma:

1. El sistema divide una de las relaciones (por ejemplo, r). Se puede utilizar en r cualquier técnica de división, incluida la división por turno rotatorio.
2. El sistema replica la otra relación, s, en todos los procesadores.
3. El procesador P_i procesa localmente la reunión de r_i con toda la relación s, utilizando cualquier técnica de reunión.

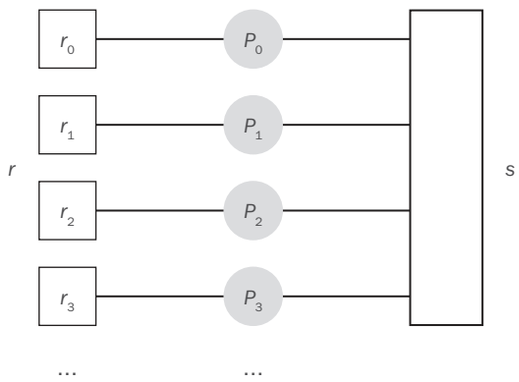
El esquema asimétrico de fragmentos y réplicas se muestra en la Figura 18.3a. Si r ya está almacenada por particiones no es necesario dividirla más en el paso 1. Todo lo que hace falta es replicar s en todos los procesadores.

El caso general de **reunión con fragmentos y réplicas** se muestra en la Figura 18.3b; funciona de la siguiente forma: el sistema divide la relación r en n particiones, $r_0, r_1, \ldots, r_{n-1}$ y s en otras m, $s_0, s_1, \ldots, s_{m-1}$. Al igual que antes, se puede utilizar cualquier técnica de división para r y para s. No es necesario que los valores de m y de n sean iguales, pero deben escogerse de modo que haya al menos $m * n$ procesadores. El esquema asimétrico de fragmentos y réplicas solo es un caso especial de fragmentos y réplicas, en el que $m = 1$. El esquema de fragmentos y réplicas reduce el tamaño de las relaciones en cada procesador en comparación con el caso asimétrico.

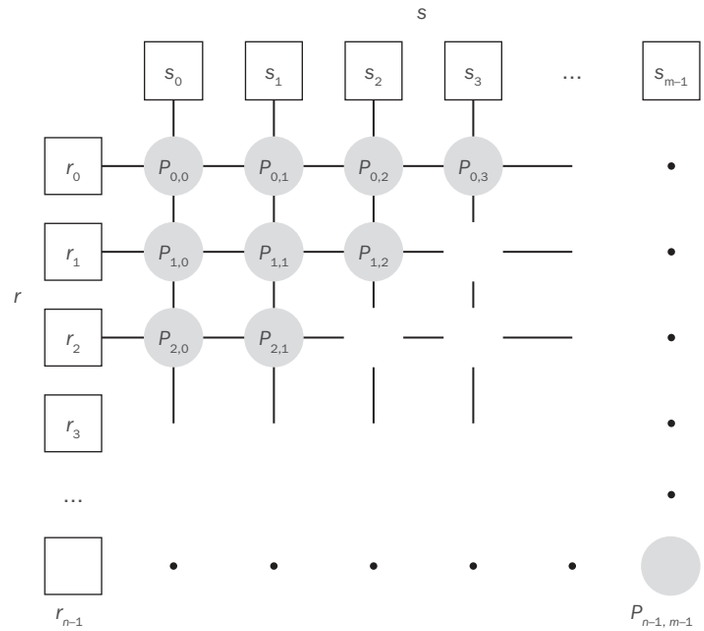
Sean los procesadores $P_{0,0}, P_{0,1}, \ldots, P_{0,m-1}, P_{1,0} \ldots, P_{n-1,m-1}$. El procesador $P_{i,j}$ procesa la reunión de r_i con s_j. Cada procesador debe obtener las tuplas de las particiones sobre las que trabaja. Para ello, el sistema replica r_i en los procesadores $P_{i,0}, P_{i,1} \ldots, P_{i,m-1}$ (que forman una fila en la Figura 18.3b) y s_i en los procesadores $P_{0,i}, P_{1,i} \ldots, P_{n-1,i}$ (que forman una columna en la Figura 18.3b). En cada procesador $P_{i,j}$ se puede utilizar la técnica de reunión que se prefiera.

El esquema de fragmentos y réplicas funciona con cualquier condición de reunión, ya que cada tupla de r puede compararse con cada una de las de s. Por tanto, puede utilizarse cuando no pueda emplearse la división.

El esquema de fragmentos y réplicas suele tener un mayor coste que la división cuando ambas relaciones son aproximadamente del mismo tamaño, ya que hay que replicar, como mínimo, una de las relaciones. Sin embargo, si una de las relaciones (por ejemplo, s) es pequeña, puede resultar más barato replicar s en todos los procesadores que volver a dividir r y s basándose en los atributos de reunión. En tal caso, el esquema asimétrico de fragmentos y réplicas es preferible, aunque pueda utilizarse la división.



(a) Fragmentos y réplicas asimétricos.



(b) Fragmentos y réplicas.

Figura 18.3. Esquemas de fragmentos y réplicas.

18.5.2.3. Reunión por asociación dividida en paralelo

La reunión por asociación dividida de la Sección 12.5.5 puede hacerse en paralelo. Suponga que se tienen n procesadores, $P_0, P_1, \ldots, P_{n-1}$ y dos relaciones, r y s, que se encuentran divididas entre varios discos. Recuerde de la Sección 12.5.5 que se debe escoger la relación de menor tamaño como relación de construcción. Si el tamaño de s es menor que el de r, el algoritmo de reunión por asociación paralela procede de la siguiente forma:

1. Se escoge una función de asociación, por ejemplo h_1, que tome el valor del atributo de reunión de cada tupla de r y de s y asigne esa tupla a uno de los n procesadores. Sean r_i las tuplas de la relación r asignadas al procesador P_i; análogamente, sean s_i las tuplas de la relación s asignadas al procesador P_i. Cada procesador P_i lee las tuplas de s que están en el disco D_i y envía cada tupla al procesador apropiado basándose en la función de asociación h_1.
2. A medida que el procesador de destino P_i recibe las tuplas de s_i, las vuelve a dividir de acuerdo con otra función de asociación, h_2, que utiliza para procesar localmente la reunión por asociación. La división en esta etapa es exactamente la misma que en la fase de división del algoritmo secuencial de reunión por asociación. Cada procesador P_i ejecuta esta fase independientemente de los demás procesadores.
3. Una vez que se han distribuido las tuplas de s, el sistema redistribuye la relación de mayor tamaño, r, entre los n procesadores de acuerdo con la función de asociación h_1 del mismo modo que anteriormente. A medida que recibe cada tupla, el procesador de destino la divide según la función h_2, igual que la relación de exploración se divide en el algoritmo secuencial de reunión por asociación.
4. Cada procesador P_i ejecuta las fases de construcción y exploración del algoritmo de reunión por asociación en las particiones locales r_i y s_i para generar una división del resultado final de la reunión por asociación.

La reunión por asociación realizada en cada procesador es independiente de las realizadas en los demás, y recibir las tuplas de r_i y de s_i es parecido a leerlas del disco. Por tanto, también se puede aplicar cualquiera de las optimizaciones de la reunión por asociación descritas en el Capítulo 12 al caso paralelo. En concreto, se puede utilizar el algoritmo híbrido de reunión por asociación para almacenar en caché algunas de las tuplas entrantes en la memoria, y evitar así el coste de escribirlas y volver a leerlas.

18.5.2.4. Reuniones con bucles anidados en paralelo

Para ilustrar el empleo de la paralelización basada en fragmentos y réplicas se considera el caso en que la relación s sea mucho menor que r. Suponga que la relación r se almacena por división; el atributo de acuerdo con el cual se divide es irrelevante. Suponga también que hay un índice basado en un atributo de reunión de la relación r en cada una de las particiones de la relación r.

Se utiliza el esquema asimétrico de fragmentos y réplicas mientras se replica la relación s y se emplea la división ya existente de la relación r. Cada procesador P_j en que se almacena una partición de la relación s lee las tuplas de la relación s almacenadas en D_j y las replica en el resto de procesadores P_i. Al final de esta fase, la relación s está replicada en todos los puntos de almacenamiento de las tuplas de la relación r.

A continuación, cada procesador P_i realiza una reunión indexada con bucles anidados de la relación s con la partición i-ésima de la relación r. Se puede solapar la reunión indexada con bucles anidados con la distribución de las tuplas de la relación s para reducir el coste de escribir en el disco las tuplas de la relación s y volver a leerlas. Sin embargo, la réplica de la relación s debe sincronizarse con la reunión para que haya espacio suficiente en las memorias intermedias de la memoria principal de cada procesador P_i para albergar las tuplas de la relación s que se hayan recibido, pero que todavía no se hayan utilizado en la reunión.

18.5.3. Otras operaciones relacionales

También se puede realizar en paralelo la evaluación de otras operaciones relacionales:

- **Selección.** Sea la selección $\sigma_\theta(r)$. Considere en primer lugar el caso en el que θ es de la forma $a_i = v$, donde a_i es un atributo y v es un valor. Si la relación r se divide de acuerdo con a_i, la selección se lleva a cabo en un solo procesador. Si θ es de la forma $l \leq a_i \leq u$ (es decir, que θ es una selección de rango) y la relación se ha dividido por rangos según a_i, entonces la selección se lleva a cabo en cada procesador cuya partición se solape con el rango de valores especificado. En el resto de los casos, la selección se lleva a cabo en todos los procesadores en paralelo.
- **Eliminación de duplicados.** Los duplicados se pueden eliminar por ordenación; para ello se puede utilizar cualquiera de las técnicas de ordenación en paralelo, optimizada para eliminar los duplicados durante la ordenación en cuanto aparezcan. También se puede paralelizar la eliminación de duplicados dividiendo las tuplas (por rangos o por asociación) y eliminado los duplicados localmente en cada procesador.
- **Proyección.** La proyección sin eliminación de duplicados se puede llevar a cabo a medida que se leen en paralelo las tuplas del disco. Si se van a eliminar los duplicados, se puede utilizar cualquiera de las técnicas que se acaban de describir.
- **Agregación.** Considere una operación de agregación. Se puede paralelizar dividiendo la relación de acuerdo con los atributos de agrupación y procesando localmente los valores de agregación en cada procesador. Se puede dividir por rangos o por asociación. Si la relación ya está dividida según los atributos de agrupación, se puede omitir la primera fase.

  Es posible reducir el coste de transferir las tuplas durante la división calculando parcialmente los valores de agregación antes de la división, al menos para las funciones de agregación utilizadas habitualmente. Considere una operación de agregación sobre la relación r, que utiliza la función de agregación **sum** en el atributo B y la agrupación basada en el atributo A. El sistema puede llevar a cabo la operación en cada procesador P_i sobre las tuplas de r almacenadas en el disco D_i. Este cálculo da lugar en cada procesador a tuplas con sumas parciales; hay una tupla en P_i para cada valor del atributo A presente en las tuplas de r almacenadas en D_i. El sistema divide el resultado de la agregación local de acuerdo con el atributo de agrupación A y vuelve a llevar a cabo la agregación (sobre las tuplas con sumas parciales) en cada procesador P_i para obtener el resultado final.

  Gracias a esta optimización no es necesario enviar tantas tuplas a los demás procesadores durante la división. Esta idea puede extenderse fácilmente a las funciones de agregación **min** y **max.** Las extensiones para las funciones de agregación **count** y **avg** se proponen al lector en el Ejercicio 18.12.

La paralelización de otras operaciones se trata en varios de los ejercicios.

18.5.4. Coste de la evaluación paralela de las operaciones

El paralelismo se obtiene dividiendo la E/S entre varios discos y el trabajo de los procesadores entre varios procesadores. Si se logra un reparto sin sobrecarga y no hay sesgo en el reparto del trabajo, las operaciones en paralelo que utilicen n procesadores tardarán $1/n$, lo que tardarían en un solo procesador. Ya se sabe cómo estimar el coste de operaciones como la reunión o la selección. El coste en tiempo del procesamiento paralelo sería entonces $1/n$, el del procesamiento secuencial de esa operación.

También hay que tener en cuenta los siguientes costes:

- Los **costes iniciales** de comenzar la operación en varios procesadores.
- El **sesgo** en la distribución del trabajo entre los procesadores, con algunos procesadores con mayor número de tuplas que otros.
- La **competencia por los recursos** —como la memoria, los discos y la red de comunicaciones— que dan lugar a retrasos.
- El **coste de construir** el resultado final mediante la transmisión de los resultados parciales desde cada procesador.

El tiempo empleado por una operación en paralelo puede estimarse como:

$$T_{\text{part}} + T_{\text{asm}} + \max(T_0, T_1, \ldots, T_{n-1})$$

donde T_{part} es el tiempo necesario para dividir las relaciones, T_{asm} es el tiempo empleado en construir los resultados y T_i el tiempo utilizado por la operación en el procesador P_i. Suponiendo que las tuplas se distribuyen sin sesgo, el número de tuplas enviadas a cada procesador puede estimarse como $1/n$ del número total de tuplas. Ignorando la competencia, el coste T_i de las operaciones en cada procesador P_i puede estimarse mediante las técnicas descritas en el Capítulo 12.

La estimación anterior es optimista, dado que es habitual que exista sesgo. Aunque dividir una sola consulta en varias fases paralelas reduce el tamaño promedio de cada parte, es el tiempo de procesamiento de la parte más lenta el que determina el tiempo empleado en procesar la consulta en su conjunto. Una evaluación en paralelo dividida no puede ser más rápida que la más lenta de sus ejecuciones en paralelo. Por tanto, cualquier sesgo en la distribución del trabajo entre los procesadores afecta mucho al rendimiento.

El problema del sesgo de la división está íntimamente relacionado con el del desbordamiento de las divisiones en las reuniones secuenciales por asociación (Capítulo 12). Se puede utilizar la resolución del desbordamiento y las técnicas de evitación desarrolladas para las reuniones por asociación para tratar el sesgo cuando se utilice la división por asociación. Se puede utilizar la división equilibrada por rangos y la división con procesadores virtuales para minimizar el sesgo debido a la división por rangos, como en la Sección 18.2.3.

18.6. Paralelismo entre operaciones

Existen dos formas de paralelismo entre operaciones: el paralelismo de encauzamiento y el paralelismo independiente.

18.6.1. Paralelismo de encauzamiento

Como se estudió en el Capítulo 12, el encauzamiento supone una importante fuente de economía de cálculo para el procesamiento de las consultas de bases de datos. Hay que recordar que, en el encauzamiento, las tuplas resultado de una operación, A, las consume una segunda operación, B, incluso antes de que la primera operación haya producido todo el conjunto de tuplas de su resultado. La ventaja principal de la ejecución encauzada de las evaluaciones secuenciales es que se puede ejecutar una secuencia de operaciones de ese tipo sin escribir en el disco ninguno de los resultados intermedios.

Los sistemas paralelos utilizan el encauzamiento principalmente por la misma razón que los sistemas secuenciales. Sin embargo, el encauzamiento también es una fuente de paralelismo, del mismo modo que el encauzamiento de instrucciones se utiliza como fuente de paralelismo en el diseño de hardware. Es posible ejecutar simultáneamente A y B en procesadores diferentes de modo que B consuma las tuplas en paralelo con su producción por A. Esta forma de paralelismo se denomina **paralelismo de encauzamiento.**

Considere una reunión de cuatro relaciones:

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$$

Se puede configurar un cauce que permita que las tres reuniones se calculen en paralelo. Suponga que se asigna al procesador P_1 el cálculo de $temp_1 \leftarrow r_1 \bowtie r_2$ y al procesador P_2 el cálculo de $r_3 \bowtie temp_1$. A medida que P_1 procesa las tuplas de $r_1 \bowtie r_2$, las pone a disposición del procesador P_2. Por tanto, P_2 tiene a su disposición algunas de las tuplas de $r_1 \bowtie r_2$ antes de que P_1 haya finalizado su cálculo. P_2 puede utilizar esas tuplas que están disponibles para comenzar el cálculo de $temp_1 \bowtie r_3$, incluso antes de que P_1 haya calculado completamente $r_1 \bowtie r_2$. Análogamente, a medida que P_2 procesa las tuplas de $(r_1 \bowtie r_2) \bowtie r_3$, las pone a disposición de P_3, que calcula su reunión con r_4.

El paralelismo encauzado resulta útil con un número pequeño de procesadores, pero no puede ampliarse bien. En primer lugar, las cadenas del cauce no suelen lograr la longitud suficiente para proporcionar un alto grado de paralelismo. En segundo lugar, no es posible encauzar los operadores relacionales que no producen resultados hasta que han accedido a todos los datos, como en el caso de la operación diferencia de conjuntos. En tercer lugar, solo se obtiene una aceleración marginal en el caso frecuente de que el coste de ejecución de un operador sea mucho mayor que el de los demás.

Por consiguiente, cuando el grado de paralelismo es elevado, el encauzamiento es una fuente de paralelismo menos importante que la división. El verdadero motivo del empleo del encauzamiento es que las ejecuciones encauzadas pueden evitar escribir en el disco los resultados intermedios.

18.6.2. Paralelismo independiente

Las operaciones en las expresiones de las consultas que son independientes entre sí pueden ejecutarse en paralelo. Esta forma de paralelismo se denomina **paralelismo independiente.**

Considérese la reunión $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$. Evidentemente, se puede procesar $temp_1 \leftarrow r_1 \bowtie r_2$ en paralelo con $temp_2 \leftarrow r_3 \bowtie r_4$. Cuando se completen esos dos cálculos, se calculará

$$temp_1 \bowtie temp_2$$

Para obtener más paralelismo se pueden encauzar las tuplas de $temp_1$ y $temp_2$ al cálculo de $temp_1 \bowtie temp_2$, que se ejecuta mediante una reunión encauzada (Sección 12.7.2.2).

Como ocurre con el paralelismo encauzado, el paralelismo independiente no proporciona un alto grado de paralelismo y resulta menos útil en sistemas con un elevado nivel de paralelismo, aunque es útil con un grado menor de paralelismo.

18.7. Optimización de consultas

Los optimizadores de consultas son responsables de gran parte del éxito de la tecnología relacional. Recuerde que los optimizadores de consultas toman una consulta y hallan el plan de ejecución de menor coste de entre todos los que proporcionan la misma respuesta.

Los optimizadores de consultas para la evaluación en paralelo de las consultas son más complicados que los correspondientes para la evaluación secuencial de consultas. En primer lugar, los modelos de costes son más complicados, ya que hay que tener en cuenta los costes de división y aspectos como el sesgo y la competencia por los recursos. Más importante aún es la manera de paralelizar las consultas. Suponga que, de algún modo, se ha escogido una expresión (de entre las equivalentes a la consulta) para utilizarla con el fin de evaluar la consulta. La expresión puede representarse mediante un árbol de operadores, como en la Sección 12.1.

Para evaluar un árbol de operadores en un sistema paralelo hay que tomar las siguientes decisiones:

- El modo de paralelizar cada operación y el número de procesadores que se emplearán para ello.
- Las operaciones que se encauzarán entre los diferentes procesadores, las operaciones que se ejecutarán independientemente en paralelo y las que lo harán secuencialmente, una tras otra.

Estas decisiones constituyen la tarea de **planificación** del árbol de ejecución.

Determinar los recursos de cada clase —como procesadores, discos y memoria— que se deben asignar a cada operación del árbol es otro aspecto del problema de la optimización. Por ejemplo, puede que parezca conveniente utilizar la máxima cantidad disponible de paralelismo, pero es buena idea no ejecutar ciertas operaciones en paralelo. Las operaciones cuyos requisitos de cálculo sean significativamente menores que la sobrecarga de comunicaciones deben agruparse con una de sus vecinas. En caso contrario, la ventaja del paralelismo se anula debido a la sobrecarga en las comunicaciones.

Un problema es que los cauces largos no se prestan a un buen empleo de los recursos. A menos que las operaciones tengan grano grueso, puede que la operación final del encauzamiento espere mucho tiempo para obtener sus datos, mientras retiene recursos preciosos, como la memoria. Por tanto, deben evitarse los cauces largos.

El número de planes de evaluación en paralelo entre los que se puede escoger es mucho mayor que el de los secuenciales. Optimizar las consultas en paralelo teniendo en cuenta todas las alternativas es mucho más costoso que optimizar las consultas secuenciales. Por tanto, para reducir el número de planes de ejecución en paralelo que se deben tomar en consideración se suelen adoptar enfoques heurísticos. A continuación se describen dos heurísticas usuales.

La primera heurística es considerar únicamente los planes de evaluación que paralelizan todas las operaciones de todos los procesadores y que no utilizan encauzamiento. Este enfoque se utiliza en las máquinas de los sistemas de Teradata. Buscar el mejor plan de ejecución de este tipo es parecido a realizar la optimización de consultas en sistemas secuenciales. Las principales diferencias radican en la manera de llevar a cabo la división y en la fórmula de estimación de costes utilizada.

La segunda heurística es escoger el plan de evaluación secuencial más eficiente y luego paralelizar sus operaciones. El sistema de bases de datos paralelas Volcano ha popularizado el modelo de paralelización denominado **intercambio de operadores.** Este modelo utiliza implementaciones ya existentes de las operaciones, que actúan sobre copias locales de los datos, acopladas con una operación de intercambio que traslada los datos entre los diferentes procesadores. Se pueden introducir los operadores de intercambio en el plan de evaluación para transformarlo en un plan de evaluación en paralelo.

Otra dimensión más de la optimización es el diseño de la organización del almacenamiento físico para acelerar las consultas. La organización física óptima es diferente para las diferentes consultas. El administrador de la base de datos debe escoger la organización física que considere adecuada para la combinación esperada de consultas a la base de datos. Por tanto, el área de la optimización de consultas en paralelo es compleja y sigue siendo un campo de investigación activa.

18.8. Diseño de sistemas paralelos

Hasta ahora, este capítulo se ha concentrado en la paralelización del almacenamiento de los datos y del procesamiento de las consultas. Dado que los sistemas de bases de datos paralelas de gran escala se utilizan principalmente para almacenar grandes volúmenes de datos y para procesar consultas de ayuda a las decisiones basadas en esos datos, estos aspectos son los más importantes de los sistemas de bases de datos paralelas. La carga de los datos en paralelo desde fuentes externas es un requisito importante si se van a tratar grandes volúmenes de datos entrantes.

Los grandes sistemas de bases de datos paralelas deben abordar también los siguientes aspectos de disponibilidad:

- La capacidad de recuperación frente a fallos de algunos procesadores o discos.
- La reorganización interactiva de los datos y los cambios interactivos de los esquemas.

Estos temas se tratan a continuación.

Con un gran número de procesadores y de discos la probabilidad de que al menos un procesador o un disco funcionen mal es significativamente mayor que en los sistemas con un único procesador y un solo disco. Un sistema paralelo mal diseñado dejará de funcionar si cualquier componente (procesador o disco) falla. Suponiendo que la probabilidad de fallo de cada procesador o disco sea pequeña, la probabilidad de fallo del sistema aumenta linealmente con el número de procesadores y de discos. Si un solo procesador o disco falla una vez cada cinco años, un sistema con cien procesadores tendrá un fallo cada dieciocho días.

Por tanto, los sistemas de bases de datos paralelos de gran escala, como Teradata y las Informix XPS de IBM, se diseñan para funcionar aunque falle un procesador o un disco. Los datos se replican en, al menos, dos procesadores. Si falla un procesador, se puede seguir accediendo desde los demás procesadores a los datos que almacenaba. El sistema hace un seguimiento de los procesadores averiados y distribuye el trabajo entre los que funcionan. Las peticiones de datos que estaban almacenados en el emplazamiento estropeado se desvían automáticamente a los emplazamientos de respaldo, que almacenan una réplica. Si todos los datos del procesador A se replican en un solo procesador B, B tendrá que procesar todas las peticiones formuladas a A, así como las propias, y eso hará que B se transforme en un cuello de botella. Por tanto, las réplicas de los datos de cada procesador se dividen entre varios procesadores.

Cuando se manejan grandes volúmenes de datos (del orden de terabytes), las operaciones sencillas, como la creación de índices, y los cambios en los esquemas, como añadir una columna a una relación, pueden tardar mucho tiempo; quizás horas o incluso días. Por tanto, no resulta aceptable que los sistemas de bases de datos no estén disponibles mientras se llevan a cabo esas operaciones. Muchos sistemas de bases de datos paralelas permiten que tales operaciones se lleven a cabo **interactivamente,** es decir, mientras el sistema ejecuta otras transacciones.

Considere, por ejemplo, la **generación interactiva de índices.** Los sistemas que tienen esta característica permiten que se realicen inserciones, borrados y actualizaciones en una relación aunque se esté generando un índice de la misma. La operación de generación de índices, por tanto, no puede bloquear toda la relación en modo compartido, como habría hecho en otro caso. Por el contrario, el proceso hace un seguimiento de las actualizaciones que tienen lugar mientras está activo e incorpora los cambios en el índice que se está generando. La mayoría de los sistemas de bases de datos actuales admiten la generación interactiva de índices, ya que esta función es muy importante incluso para sistemas de bases de datos no paralelas.

En los últimos años, un cierto número de compañías han desarrollado nuevos productos de bases de datos paralelas, entre ellos Netezza, DATAllegro (adquirida por Microsoft) Greenplum y Aster Data. Cada uno de estos productos se ejecuta en sistemas que constan de decenas a miles de nodos, cada nodo ejecutando una instancia de la base de datos. Cada producto maneja la división de datos, así como el procesamiento paralelo de consultas, en todas las instancias de la base de datos.

Netezza, Greenplum y Aster Data usan PostgreSQL como base de datos subyacente; DATAllegro usaba inicialmente Ingres, pero cambió a SQL Server tras su adquisición por Microsoft. Al construirse encima de un sistema de bases de datos existente, estos sistemas pueden aprovechar las funciones de almacenamiento de datos, procesamiento de consultas y gestión de transacciones de la base de datos subyacente, para centrarse en la división de los datos (incluyendo la replicación para la tolerancia a fallos), la comunicación entre procesos rápida, el procesamiento paralelo de consultas y la optimización de consultas paralelas. Otra ventaja de utilizar una base de datos de dominio público como base de datos, como PostgreSQL, es que el coste del software por nodo es muy bajo, al contrario que las bases de datos comerciales, que tienen un coste significativo por procesador.

También es importante mencionar que Netezza y DATAllegro realmente venden «dispositivos» data warehouse, que incluyen hardware y software, lo que permite que los clientes construyan bases de datos paralelas con un mínimo esfuerzo.

18.9. Paralelismo en procesadores multinúcleo

El paralelismo se ha convertido en un elemento común de las computadoras actuales, incluso en algunas de las más pequeñas, debido a las tendencias en arquitectura de computadoras. Por ello, todas las bases de datos actuales se ejecutan, virtualmente, en una plataforma paralela. En esta sección se tratarán brevemente las razones de esta tendencia en la arquitectura de computadoras y los efectos que tiene sobre el diseño e implementación de las bases de datos.

18.9.1. Paralelismo frente a velocidad en bruto

Desde la aparición de las computadoras, la velocidad de los procesadores ha ido creciendo a un ritmo exponencial, doblándose cada 18 a 24 meses. Este incremento supone un crecimiento exponencial en el número de transistores que pueden ponerse por unidad de área en un chip de silicio, que se conoce popularmente como **ley de Moore,** que toma el nombre del cofundador de Intel, Gordon Moore. Técnicamente, la ley de Moore no es una *ley* sino una observación y una predicción de las tendencias tecnológicas. Hasta

hace poco tiempo, el crecimiento en el número de transistores y la reducción de su tamaño ha permitido la creación de procesadores más rápidos. Aunque el progreso tecnológico continúa cumpliendo la predicción de la ley de Moore, ha aparecido otro factor que frena el crecimiento de la velocidad de los procesadores. Los procesadores más rápidos son energéticamente ineficientes. Es un problema en cuanto a la energía consumida y en coste, vida de las baterías para la computación móvil y disipación de calor (eventualmente toda la energía consumida se convierte en calor). Por ello, los procesadores modernos normalmente no tienen un único procesador sino que suelen constar de varios procesadores en un chip. Para establecer una distinción entre los multiprocesadores en un chip y los procesadores tradicionales se utiliza el término **núcleo** para los procesadores en un chip. En este sentido se dice que una máquina tiene un procesador multinúcleo.[2]

18.9.2. Memoria caché y multihilos

Cada núcleo puede procesar un flujo independiente de instrucciones de máquina. Sin embargo, como los procesadores pueden procesar los datos más rápido de lo que pueden acceder a los mismos en la memoria principal, esta se convierte en un cuello de botella que limita el rendimiento global. Por esta razón, los diseñadores de computadoras incluyeron uno o más niveles de memoria **caché** en el sistema. La memoria caché es más cara que la memoria principal en coste por byte, pero ofrece un acceso más rápido. En los diseños de caché multinivel, los niveles se denominan L1, L2, etc., siendo L1 la caché más rápida, y por tanto la de mayor coste por byte y además la más pequeña, L2 la siguiente en velocidad, y así sucesivamente. Supone una extensión del almacenamiento jerárquico que se trató en el Capítulo 10 para incluir los distintos niveles de caché por debajo de la memoria principal.

Aunque el sistema de bases de datos puede controlar la transferencia de datos entre el disco y la memoria principal, el hardware de la computadora tiene el control de la transferencia de los datos entre los distintos niveles de la caché y entre la caché y la memoria principal. A pesar de esta falta de control directo, el rendimiento del sistema de bases de datos se puede ver afectado por cómo se utiliza la caché. Si un núcleo necesita acceder a un elemento de datos que no está en caché, debe localizarse en la memoria principal. Como la memoria principal es mucho más lenta que los procesadores, se puede perder una considerable capacidad de procesamiento mientras el núcleo espera el dato desde la memoria principal. Estas esperas se denominan **pérdidas de caché.**

Una forma en la que los diseñadores de computadoras intentan mitigar el impacto de las pérdidas de caché es mediante *multihilos*. Un **hilo** es un flujo de ejecución que comparte memoria[3] con otros hilos que se ejecutan en el mismo núcleo. Si el hilo que se está ejecutando en un núcleo sufre una pérdida de caché, u otro tipo de espera, el núcleo pasa a ejecutar otro hilo, no desperdiciando la capacidad de cómputo mientras espera.

Los hilos introducen otra fuente de paralelismo distinta a la de la multiplicidad de los núcleos. Cada nueva generación de procesadores admite más núcleos y más hilos. El procesador Sun UltraSPARC T2 tiene 8 núcleos, cada uno de ellos admite hasta 8 hilos, hasta un total de 64 hilos en un chip procesador.

La tendencia en arquitecturas de computadores de un menor incremento en la velocidad en bruto acompañada por un crecimiento en el número de núcleos tiene implicaciones importantes en el diseño de bases de datos, como se verá en breve.

2 El uso del término *núcleo* tiene un significado diferente de aquel que se utilizaba en los primeros días de la computación para referirse a los núcleos magnéticos en que se basaba la tecnología de memoria principal.

3 Técnicamente, en la terminología de los sistemas de bases de datos, su espacio de direcciones.

18.9.3. Adaptación del diseño de sistemas de bases de datos a las modernas arquitecturas de computadoras

Podría parecer que los sistemas de bases de datos son una aplicación ideal para aprovechar el gran número de núcleos e hilos, ya que los sistemas de bases de datos permiten un gran número de transacciones concurrentes. Sin embargo, existen toda una variedad de factores que convierten en un desafío hacer un uso óptimo de los procesadores modernos.

Cuando se permite un mayor grado de concurrencia para aprovechar las ventajas del paralelismo en los procesadores modernos, se aumenta la cantidad de datos que se necesita en la memoria caché. Esto genera mayores pérdidas de caché, quizá demasiadas para que incluso un núcleo multihilo tenga que esperar por los datos desde la memoria principal.

Las transacciones concurrentes necesitan cierta forma de control de concurrencia para asegurar las propiedades ACID que se vieron en el Capítulo 14. Cuando transacciones concurrentes acceden a datos comunes, se deben imponer ciertas restricciones de acceso en el acceso concurrente. Estas restricciones, basadas en bloqueos, marcas de tiempo o validación, generan esperas o pérdidas de trabajo debido a que las transacciones pueden quedar abortadas. Para evitar esperas excesivas o trabajo perdido, lo ideal es que las transacciones entren en conflicto en raras ocasiones, pero intentar asegurarlo puede aumentar la cantidad de datos que se necesita en la caché, lo que genera más pérdidas de caché.

Por último, existen componentes del sistema de bases de datos que comparten todas las transacciones. En un sistema que utiliza bloqueos, la tabla de bloqueos se comparte por todas las transacciones y el acceso a la misma puede suponer un cuello de botella. Existen problemas parecidos con otras formas de control de concurrencia. De forma similar, el gestor de memoria intermedia, el gestor del registro histórico y el gestor de recuperaciones sirven a todas las transacciones y son potenciales cuellos de botella.

Como disponer de un gran número de transacciones concurrentes puede no aprovechar las ventajas de los procesadores modernos, es deseable encontrar formas para que varios núcleos trabajen en una única transacción. Esto requiere que el procesador de consultas de la base de datos encuentre formas eficaces de paralelizar las consultas sin crear una demanda excesiva sobre la caché. Se puede hacer creando cauces con las operaciones de consultas de la base de datos y encontrando formas de paralelizar las operaciones individuales.

La adaptación del diseño de sistemas de bases de datos y del procesamiento de consultas a los sistemas multinúcleo y multihilo sigue siendo un área de investigación activa. Consulte las notas bibliográficas para más información.

18.10. Resumen

- Las bases de datos paralelas han logrado una aceptación comercial significativa en los últimos veinte años.
- En el paralelismo de E/S, las relaciones se dividen entre los discos disponibles para poder recuperarlas más rápidamente. Tres técnicas de división utilizadas frecuentemente son la división por turno rotatorio, la división por asociación y la división por rangos.
- El sesgo es un problema importante, especialmente con grados elevados de paralelismo. Los vectores de división equilibrados, que utilizan histogramas, y la división con procesadores virtuales son algunas técnicas usadas para reducir el sesgo.
- En el paralelismo entre consultas se ejecutan concurrentemente diferentes consultas para aumentar la productividad.

- El paralelismo en las consultas intenta reducir el coste de ejecución de las consultas. Existen dos tipos de paralelismo en las consultas: el paralelismo en operaciones y el paralelismo entre operaciones.
- El paralelismo en operaciones se utiliza para ejecutar operaciones relacionales, como las ordenaciones y las reuniones, en paralelo. El paralelismo en operaciones es algo natural en las operaciones relacionales, ya que están orientadas a conjuntos.
- Existen dos enfoques básicos en la paralelización de las operaciones binarias como las reuniones.
  - En el paralelismo de divisiones, las relaciones se dividen en varias partes y las tuplas de r_i solo se reúnen con las tuplas de s_i. El paralelismo de divisiones solo se puede usar para las reuniones naturales y para las equirreuniones.
  - En el esquema de fragmentos y réplicas, las dos relaciones se dividen y cada división se duplica. En el esquema asimétrico de fragmentos y réplicas, una de las relaciones se replica mientras la otra se divide. A diferencia del paralelismo de divisiones, el esquema de fragmentos y réplicas, simétrico o asimétrico, puede utilizarse con cualquier condición de reunión.

    Ambas técnicas de paralelismo pueden utilizarse en combinación con cualquiera de las técnicas de reunión.
- En el paralelismo independiente, las diferentes operaciones que son independientes entre sí se ejecutan en paralelo.
- En el paralelismo encauzado, los procesadores envían los resultados de una operación a otra a medida que los van calculando, sin esperar a que concluya toda la operación.
- La optimización de consultas en las bases de datos paralelas es significativamente más compleja que su equivalente en las bases de datos secuenciales.
- Los modernos procesadores multinúcleo están generando nuevos problemas de investigación en las bases de datos paralelas.

Términos de repaso

- Consultas de ayuda a la toma de decisiones.
- Paralelismo de E/S.
- División horizontal.
- Técnicas de división.
  - Turno rotatorio.
  - División por asociación.
  - División por rangos.
- Atributo de división.
- Vector de división.
- Consulta concreta.
- Consulta de rango.
- Sesgo.
  - De ejecución.
  - De los valores de los atributos.
  - De la división.
- Manejo del sesgo.
  - Vector de división por rangos equilibrado.
  - Histograma.
  - Procesadores virtuales.
- Paralelismo entre consultas.
- Coherencia de la caché.
- Paralelismo en consultas.
  - Paralelismo en operaciones.
  - Paralelismo entre operaciones.
- Ordenación paralela.
  - Ordenación con división por rangos.
  - Ordenación y mezcla externas paralelas.
- Paralelismo de datos.
- Reunión paralela.
  - Por división.
  - Con fragmentos y réplicas.
  - Con esquema asimétrico de fragmentos y réplicas.
  - Por asociación dividida en paralelo.
  - Con bucles anidados en paralelo.
- Selección paralela.
- Eliminación de duplicados en paralelo.
- Proyección paralela.
- Agregación paralela.
- Coste de la evaluación paralela.
- Paralelismo entre operaciones.
  - Paralelismo de encauzamiento.
  - Paralelismo independiente.
- Optimización de consultas.
- Planificación.
- Modelo del operador de intercambio.
- Diseño de sistemas paralelos.
- Creación interactiva de índices.
- Procesadores multinúcleo.

Ejercicios prácticos

18.1. En una selección de rango sobre un atributo dividido por rangos es posible que solo haga falta acceder a un disco.

Describa las ventajas e inconvenientes de esta propiedad.

18.2. Indique la forma de paralelismo (entre consultas, entre operaciones o en operaciones) que puede resultar más importante para cada una de las tareas siguientes:

a. Incrementar el rendimiento de un sistema con muchas consultas pequeñas.

b. Incrementar el rendimiento de un sistema con unas pocas consultas de gran tamaño cuando el número de discos y de procesadores es elevado.

18.3. Con el paralelismo de encauzamiento suele resultar conveniente llevar a cabo varias operaciones de un cauce en un mismo procesador, aunque haya disponibles varios procesadores.

a. Explique el motivo.

b. ¿Serían válidos los argumentos defendidos en el apartado *a* si la máquina tuviera una arquitectura de memoria compartida? Explique por qué o por qué no.

c. ¿Serían válidos los argumentos del apartado *a* con paralelismo independiente? Es decir, ¿hay casos en que, incluso si las operaciones no se encauzan y hay muchos procesadores disponibles, sigue siendo conveniente llevar a cabo varias operaciones en el mismo procesador?

18.4. Considere el procesamiento de reuniones utilizando el esquema simétrico de fragmentos y réplicas con división por rangos. ¿Cómo se puede optimizar la evaluación si la condición de reunión es de la forma $|r.A - s.B| \leq k$, donde k es una constante pequeña? Aquí, $|x|$ indica el valor absoluto de x. Las reuniones con una condición de reunión así se denominan **reuniones de banda.**

18.5. Recuerde que los histogramas se utilizan para generar particiones de rangos con carga equilibrada.

a. Suponga que se tiene un histograma en el que los valores varían de 1 a 100 y están divididos en 10 intervalos, 1–10, 11–20 …, 91–100, con las frecuencias 15, 5, 20, 10, 10, 5, 5, 20, 5 y 5, respectivamente. Proponga una función de división por rangos con carga equilibrada para dividir los valores en cinco particiones.

b. Escriba un algoritmo para calcular una división por rangos con carga equilibrada con p particiones, dado un histograma de las distribuciones de frecuencias que contenga n rangos.

18.6. Los sistemas de bases de datos paralelos de gran escala almacenan una copia adicional de cada elemento de los datos en discos conectados a un procesador diferente, para evitar la pérdida de los datos si falla alguno de los procesadores.

a. En lugar de guardar una copia adicional de los elementos de datos de un procesador a un único procesador de copia de seguridad, es una buena idea dividir las copias de los elementos de datos de un procesador entre varios procesadores. Explique por qué.

b. Explique cómo se puede usar la división de procesadores virtuales para implementar una división eficiente de las copias como se ha descrito anteriormente.

c. ¿Cuáles son las ventajas e inconvenientes de utilizar almacenamiento RAID en lugar de almacenar una copia adicional de cada elemento de datos?

18.7. Suponga que se desea indexar una relación grande que está dividida. ¿Se puede aplicar la idea de la división, incluyendo la división en procesadores virtuales, a los índices?

Justifique su respuesta, considerando estos dos casos (suponga por simplicidad que la división así como la indexación es sobre un único atributo):

a. Si el índice se encuentra en el atributo de división de la relación.

b. Si el índice se encuentra en un atributo distinto del de división de la relación.

18.8. Suponga que se ha elegido un vector de división de rango bien equilibrado para una relación, pero que la relación se continúa actualizando, desequilibrando la división. Incluso aunque se use una división de procesadores virtuales, un procesador virtual puede acabar con un gran número de tuplas tras la actualización y se debería realizar una nueva división.

a. Suponga que un procesador virtual tiene un exceso significativo de tuplas (digamos el doble del promedio). Explique cómo se debería realizar la nueva división repartiendo, y por tanto aumentando, el número de procesadores virtuales.

b. Suponga que, en lugar de asignación por turno rotatorio de los procesadores virtuales, se asignan las divisiones a los procesadores de manera arbitraria con una tabla de asociación que controle la misma. Si un nodo dado tiene un exceso de carga (comparado con el resto), explique cómo se puede equilibrar la carga.

c. Suponiendo que no existan actualizaciones, ¿hay que detener el procesamiento de consultas mientras se realiza la nueva división o reasignación de procesadores virtuales?

Justifique su respuesta.

Ejercicios

18.9. Para cada una de las tres técnicas de división, es decir, por turno rotatorio, por asociación y por rangos, indique un ejemplo de consulta para la que esa técnica de división proporcione la respuesta más rápida.

18.10. Indique los factores que pueden dar lugar a un sesgo cuando se divide una relación de acuerdo con uno de sus atributos utilizando:

a. División por asociación.

b. División por rangos.

En cada caso, indique lo que se puede hacer para reducir el sesgo.

18.11. Proponga un ejemplo de reunión, que no sea una equirreunión simple, para la que pueda utilizarse paralelismo de divisiones. ¿Qué atributos deberían utilizarse para la división?

18.12. Describa una buena forma de paralelizar lo siguiente:

a. La operación diferencia.

b. La agregación utilizando la operación **count.**

c. La agregación utilizando la operación **count distinct.**

d. La agregación utilizando la operación **avg.**

e. La reunión externa por la izquierda, si la condición de reunión solo implica igualdad.

f. La reunión externa por la izquierda, si la condición de reunión implica comparaciones distintas de la igualdad.

g. La reunión externa completa, si la condición de reunión implica comparaciones distintas de la igualdad.

18.13. Describa las ventajas e inconvenientes del paralelismo de encauzamiento.

18.14. Suponga que se desea tratar una carga de trabajo consistente en gran cantidad de transacciones de pequeño tamaño mediante paralelismo sin compartición.

a. ¿Se necesita paralelismo en consultas en esta situación? En caso de no ser así, indique el motivo y la forma de paralelismo que se considera adecuada.

b. ¿Qué forma de sesgo sería relevante con esta carga de trabajo?

c. Suponga que la mayoría de las transacciones accediesen a un registro de *cuenta*, que incluye un atributo *tipo_cuenta*, y a un registro asociado *maestro_tipo_cuenta*, que proporciona la información sobre el tipo de cuenta. ¿Cómo se dividirían y/o duplicarían los datos para acelerar las transacciones? Se puede suponer que la relación *maestro_tipo_cuenta* se actualiza rara vez.

18.15. El atributo por el que se divide una relación puede tener un impacto significativo en el coste de la consulta.

a. Dada una carga de trabajo de consultas de SQL en una única relación, ¿qué atributos serían candidatos para la división?

b. ¿Cómo se podría seleccionar entre las técnicas de división alternativas, de acuerdo con la carga de trabajo?

c. ¿Se puede dividir una relación por más de un atributo? Justifique su respuesta.

Notas bibliográficas

A finales de los setenta y comienzos de los ochenta, a medida que el modelo relacional lograba un fundamento razonablemente sólido, se fue reconociendo que los operadores relacionales se pueden paralelizar en gran medida y que tienen buenas propiedades de flujo de datos. Se lanzaron varios proyectos de investigación como GAMMA DeWitt [1990], XPRS (Stonebraker et ál. [1989]) y Volcano (Graefe [1990]) para estudiar la viabilidad de la ejecución en paralelo de los operadores relacionales.

Teradata fue uno de los primeros sistemas de bases de datos paralelas comerciales y continúa teniendo una gran parte del mercado. El Red Brick Warehouse fue otro de los primeros sistemas de bases de datos paralelos. Red Brick fue adquirido por Informix, que a su vez fue adquirido por IBM. Otros sistemas de bases de datos paralelos recientes incluyen Netezza, DATAllegro (ahora parte de Microsoft) Greenplum y Aster Data.

El bloqueo en las bases de datos paralelas se estudia en Joshi [1991] y Mohan y Narang [1992]. Los protocolos de coherencia de caché para los sistemas paralelos de bases de datos se estudian en Dias et ál. [1989], Mohan y Narang [1992] y Rahm [1993]. Carey et ál. [1991] estudian los aspectos de la caché en los sistemas cliente-servidor.

Graefe y McKenna [1993b] presentan un excelente resumen del procesamiento de consultas, incluido el procesamiento de consultas en paralelo. Graefe [1990] y Graefe y McKenna [1993b] defendieron el modelo del operador de intercambio.

La ordenación en paralelo se estudia en DeWitt et ál. [1992]. Los algoritmos de ordenación en paralelo en procesadores multinúcleo y multihilo se estudian en García y Korth [2005] y Chen et ál. [2007]. Los algoritmos de reunión en paralelo se describen en Nakayama et ál. [1984], Richardson et ál. [1987], Kitsuregawa y Ogawa [1990] y Wilschut et ál. [1995], entre otros trabajos.

El tratamiento del sesgo en las reuniones en paralelo se describe en Walton et ál. [1991], Wolf [1991] y DeWitt et ál. [1992].

Las técnicas de optimización de consultas en paralelo se describen en Lu et ál. [1991] y Ganguly et ál. [1992].

La adaptación del diseño de los sistemas de bases de datos y los algoritmos de procesamiento de consultas a las arquitecturas multinúcleo y multihilo se estudia en Proceedings of the International Workshop on Data Management on Modern Hardware (DaMoN), que se celebra anualmente desde 2005.

Bases de datos distribuidas

19

A diferencia de los sistemas paralelos, en los que los procesadores se hallan estrechamente acoplados y constituyen un solo sistema de bases de datos, los sistemas de bases de datos distribuidos están formados por sitios débilmente acoplados que no comparten ningún componente físico. Además, puede que los sistemas de bases de datos que se ejecutan en cada sitio sean sustancialmente independientes entre sí. La estructura básica de los sistemas distribuidos se estudió en el Capítulo 17.

Cada sitio puede participar en la ejecución de transacciones que acceden a los datos de uno o de varios sitios diferentes. La diferencia principal entre los sistemas de bases de datos centralizados y los distribuidos es que, en los primeros, los datos residen en una única ubicación, mientras que en los segundos los datos se reparten entre varios lugares. Esta distribución de los datos provoca muchas dificultades en el procesamiento de las transacciones y de las consultas. En este capítulo se abordarán esas dificultades.

Se comenzará por clasificar las bases de datos distribuidas en homogéneas y heterogéneas, en la Sección 19.1. A continuación se abordará el problema del almacenamiento de los datos en las bases de datos distribuidas, en la Sección 19.2. En la Sección 19.3 se esboza un modelo de procesamiento de transacciones en las bases de datos distribuidas. En la Sección 19.4 se describe la manera de implementar transacciones atómicas en bases de datos distribuidas mediante protocolos de compromiso especiales. En la Sección 19.5 se describe el control de concurrencia en las bases de datos distribuidas. En la Sección 19.6 se esboza el modo de proporcionar una elevada disponibilidad en bases de datos distribuidas aprovechando las réplicas, de manera que el sistema pueda continuar procesando las transacciones aunque se produzca un fallo. El procesamiento de las consultas en las bases de datos distribuidas se aborda en la Sección 19.7. En la Sección 19.8 se esbozan aspectos del manejo de bases de datos heterogéneas. La Sección 19.10 describe los sistemas de directorio, que pueden considerarse una forma especializada de las bases de datos distribuidas.

En ese capítulo se ilustran todos los ejemplos usando la base de datos bancaria de la Figura 19.1.

19.1. Bases de datos homogéneas y heterogéneas

En los sistemas de **bases de datos distribuidas homogéneas** todos los sitios emplean idéntico software de gestión de bases de datos, son conscientes de la existencia de los demás sitios y acuerdan cooperar en el procesamiento de las solicitudes de los usuarios. En estos sistemas, los sitios locales renuncian a una parte de su autonomía en cuanto a su derecho a modificar los esquemas o el software de gestión de bases de datos. Ese software también debe cooperar con los demás sitios en el intercambio de la información sobre las transacciones para hacer posible su procesamiento entre varios sitios.

A diferencia de lo anterior, en las **bases de datos distribuidas heterogéneas** puede que los diferentes sitios utilicen esquemas y software de gestión de sistemas de bases de datos diferentes. Es posible que algunos sitios no tengan información de la existencia del resto y que solo proporcionen facilidades limitadas para la cooperación en el procesamiento de las transacciones. Las diferencias en los esquemas suelen constituir un problema importante para el procesamiento de las consultas, mientras que la divergencia del software supone un inconveniente para el procesamiento de transacciones que acceden a varios sitios.

Este capítulo se centrará en las bases de datos distribuidas homogéneas. No obstante, en la Sección 19.8 se estudiarán brevemente los aspectos del procesamiento de las consultas en los sistemas de bases de datos distribuidas heterogéneas.

sucursal(nombre_sucursal, ciudad_sucursal, activos)
cuenta(número_cuenta, nombre_sucursal, saldo)
depositante(nombre_cliente, número_cuenta)

Figura 19.1. Base de datos bancaria.

19.2. Almacenamiento distribuido de datos

Considere una relación r que hay que almacenar en la base de datos. Existen dos enfoques del almacenamiento de esta relación en la base de datos distribuida:

- **Réplica.** El sistema conserva varias réplicas (copias) idénticas de la relación y guarda cada réplica en un sitio diferente. La alternativa a las réplicas es almacenar solo una copia de la relación r.
- **Fragmentación.** El sistema divide la relación en varios fragmentos y guarda cada fragmento en un sitio diferente.

La fragmentación y la réplica pueden combinarse: las relaciones pueden dividirse en varios fragmentos y puede haber varias réplicas de cada fragmento. En las siguientes secciones se profundizará en cada una de estas técnicas.

19.2.1. Réplica de datos

Si la relación r se replica, se guarda una copia de esa relación en dos o más sitios. En el caso más extremo se tiene una **réplica completa,** en la que se guarda una copia en cada sitio del sistema.

Las réplicas presentan varias ventajas e inconvenientes.

- **Disponibilidad.** Si alguno de los sitios que contiene la relación *r* falla, esa relación puede hallarse en otro sitio distinto. Por tanto, el sistema puede seguir procesando las consultas que impliquen a *r*, pese al fallo del sitio.
- **Paralelismo incrementado.** En el caso de que la mayoría de los accesos a la relación *r* solo fuesen lecturas, estas se podrían procesar en paralelo desde los diferentes sitios. Cuantas más réplicas de *r* existan, mayor será la posibilidad de que los datos necesarios se encuentren en el sitio en que se ejecuta la transacción. Por tanto, la réplica de los datos minimiza su transmisión entre los diferentes sitios.
- **Sobrecarga incrementada durante la actualización.** El sistema debe asegurar que todas las réplicas de la relación *r* sean consistentes; en caso contrario pueden producirse cálculos erróneos. Por tanto, siempre que se actualiza *r* hay que propagar la actualización a todos los sitios que contienen réplicas. El resultado es una sobrecarga incrementada. Por ejemplo, en un sistema bancario, en el que la información de las cuentas se replica en varios sitios, es necesario asegurarse de que el saldo de todas las cuentas concuerde en todos ellos.

En general, la réplica mejora el rendimiento de las operaciones de lectura y aumenta la disponibilidad de los datos para las transacciones de lectura. Sin embargo, las transacciones de actualización suponen una mayor sobrecarga. El control de las actualizaciones concurrentes de los datos replicados realizadas por varias transacciones resulta más complejo que en los sistemas centralizados, los cuales se estudiaron en el Capítulo 15. Se puede simplificar la gestión de las réplicas de la relación *r* escogiendo una de ellas como **copia principal** de *r*. Por ejemplo, en un sistema bancario, las cuentas pueden asociarse con el sitio en que se abrieron. De manera parecida, en un sistema de reserva de billetes de avión, cada vuelo puede asociarse con el sitio en que se origina. El esquema de copias principales y otras opciones del control de concurrencia distribuida se examinarán en la Sección 19.5.

19.2.2. Fragmentación de los datos

Si la relación *r* se fragmenta, *r* se divide en varios *fragmentos* r_1, r_2 …, r_n. Estos fragmentos contienen suficiente información como para permitir la reconstrucción de la relación original *r*. Existen dos esquemas diferentes de fragmentación de las relaciones: la fragmentación *horizontal* y la *vertical*. La fragmentación horizontal divide la relación asignando cada tupla de *r* a uno o más fragmentos. La fragmentación vertical divide la relación descomponiendo el esquema *R* de la relación *r*.

En la **fragmentación horizontal** la relación *r* se divide en varios subconjuntos, r_1, r_2, …, r_n. Cada tupla de la relación *r* debe pertenecer, como mínimo, a uno de los fragmentos, de modo que se pueda reconstruir la relación original, si fuera necesario.

A modo de ejemplo, la relación *cuenta* se puede dividir en varios fragmentos, cada uno de los cuales consiste en tuplas de cuentas pertenecientes a una sucursal concreta. Si el sistema bancario solo tiene dos sucursales (Hillside y Valleyview), habrá dos fragmentos diferentes:

$$cuenta_1 = \sigma_{nombre_sucursal\ =\ \text{«Hillside»}}\ (cuenta)$$
$$cuenta_2 = \sigma_{nombre_sucursal\ =\ \text{«Valleyview»}}\ (cuenta)$$

La fragmentación horizontal suele emplearse para conservar las tuplas en los sitios en que más se utilizan, para minimizar la transferencia de datos.

En general, un fragmento horizontal se puede definir como una *selección* de la relación global *r*. Es decir, se utiliza un predicado P_i para construir el fragmento r_i:

$$r_i = \sigma_{P_i}\ (r)$$

La relación *r* se reconstruye tomando la unión de todos los fragmentos; es decir:

$$r = r_1 \cup r_2 \cup \ldots \cup r_n$$

En el ejemplo, los fragmentos son disjuntos. Al cambiar los predicados de selección empleados para crear los fragmentos se puede hacer que una tupla de *r* dada aparezca en más de uno de los fragmentos r_i.

En su forma más sencilla, la fragmentación vertical es igual que la descomposición (véase el Capítulo 8). La **fragmentación vertical** de *r(R)* implica la definición de varios subconjuntos de atributos $R_1, R_2 \ldots, R_n$ del esquema *R*, de modo que:

$$R = R_1 \cup R_2 \cup \ldots \cup R_n$$

Cada fragmento r_i de *r* se define mediante:

$$r_i = \Pi_{Ri}\ (r)$$

La fragmentación debe hacerse de modo que se pueda reconstruir la relación *r* a partir de los fragmentos tomando la reunión natural:

$$r = r_1 \bowtie r_2 \bowtie r_3 \bowtie \ldots \bowtie r_n$$

Una manera de asegurar que la relación *r* pueda reconstruirse es incluir los atributos de la clave principal de *R* en cada uno de los fragmentos R_i. De manera más general, se puede utilizar cualquier superclave. Suele resultar conveniente añadir un atributo especial, denominado *id_tupla*, al esquema *R*. El valor id_tupla de cada tupla es un valor único que distingue a esa tupla de todas las demás. El atributo id_tupla, por tanto, sirve como clave candidata para el esquema aumentado y se incluye en cada uno de los fragmentos R_i. La dirección física o lógica de la tupla puede utilizarse como id_tupla, ya que cada tupla tiene una dirección única.

Para ilustrar la fragmentación vertical considere una base de datos universitaria con una relación *info_empleado* que almacena, para cada empleado, *id_empleado*, *nombre*, *puesto* y *sueldo*. Por motivos de protección de la intimidad, puede que esta relación se fragmente en una denominada *empleado_infoprivada*, que contenga *id_empleado* y *sueldo*, y en otra llamada *empleado_infopública*, que contenga los atributos *id_empleado*, *nombre* y *puesto*. Puede que las dos relaciones se almacenen en sitios diferentes, nuevamente, por motivos de seguridad.

Se pueden aplicar los dos tipos de fragmentación a un mismo esquema; por ejemplo, los fragmentos obtenidos de la fragmentación horizontal de una relación pueden dividirse nuevamente de manera vertical. Los fragmentos también pueden replicarse. En general, los fragmentos pueden replicarse, las réplicas de los fragmentos pueden fragmentarse más, y así sucesivamente.

19.2.3. Transparencia

No se debe exigir a los usuarios de los sistemas distribuidos de bases de datos que conozcan la ubicación física de los datos ni el modo en que se puede acceder a ellos en cada sitio local concreto. Esta característica, denominada **transparencia de los datos,** puede adoptar varias formas:

- **Transparencia de la fragmentación.** No se exige a los usuarios que conozcan el modo en que se ha fragmentado la relación.
- **Transparencia de la réplica.** Los usuarios ven cada objeto de datos como lógicamente único. Puede que el sistema distribuido replique los objetos para incrementar el rendimiento del sistema o la disponibilidad de los datos. Los usuarios no deben preocuparse por qué objetos se han replicado ni por la ubicación de esas réplicas.
- **Transparencia de la ubicación.** No se exige a los usuarios que conozcan la ubicación física de los datos. El sistema distribuido de bases de datos debe poder hallar los datos siempre que la transacción del usuario facilite el identificador de esos datos.

Los elementos de datos (como las relaciones, los fragmentos y las réplicas) deben tener nombres únicos. Esta propiedad es fácil de asegurar en las bases de datos centralizadas. En las bases de datos distribuidas, sin embargo, hay que tener cuidado para asegurarse de que dos sitios no utilicen el mismo nombre para elementos de datos diferentes.

Una solución a este problema es exigir que todos los nombres se registren en un **servidor de nombres** central. El servidor de nombres ayuda a garantizar que el mismo nombre no se utilice para elementos de datos diferentes. También se puede utilizar el servidor de nombres para localizar los elementos de datos, dado su nombre. Este enfoque, sin embargo, presenta dos inconvenientes principales. En primer lugar, puede que el servidor de nombres se transforme en un cuello de botella para el rendimiento cuando los elementos de datos se buscan por el nombre, lo que da lugar a un bajo rendimiento. En segundo lugar, si el servidor de nombres queda fuera de servicio, puede que ningún otro sitio del sistema distribuido logre seguir en funcionamiento.

Un enfoque alternativo más utilizado exige que cada sitio anteponga su propio identificador de sitio a cualquier nombre que genere. Este enfoque garantiza que dos sitios diferentes no generen nunca el mismo nombre (dado que cada sitio tiene un identificador único). Además, no se necesita ningún control centralizado. Esta solución, no obstante, no logra conseguir transparencia en la ubicación, dado que a los nombres se les adjuntan los identificadores de los sitios. Así, se puede hacer referencia a la relación *cuenta* como *cuenta.sitio*17, o *cuenta@sitio*17, en lugar de meramente como *cuenta*. Muchos sistemas de bases de datos utilizan la dirección de Internet (dirección IP) de los sitios para identificarlos.

Para solventar este problema, el sistema de bases de datos puede crear un conjunto de nombres alternativos, o **alias,** para los elementos de datos. En consecuencia, los usuarios se pueden referir a los elementos de datos mediante nombres sencillos que el sistema traduce a los nombres completos. La relación entre los alias y los nombres reales puede almacenarse en todos los sitios. Cuando se emplean alias, el usuario puede ignorar la ubicación física de los elementos de datos. Además, no se ve afectado si el administrador de la base de datos decide trasladar un elemento de datos de un sitio a otro.

Los usuarios no deberían tener necesidad de hacer referencia a una réplica concreta de un elemento de datos. En vez de eso, el sistema debe determinar la réplica a la que hay que hacer referencia en las solicitudes leer, y actualizar todas las réplicas en las solicitudes escribir. Se puede asegurar que lo hace si se mantiene una tabla de catálogo, que el sistema utilizará para determinar todas las réplicas del elemento de datos.

19.3. Transacciones distribuidas

El acceso a los diferentes elementos de datos en los sistemas distribuidos suele realizarse mediante transacciones, que deben mantener las propiedades ACID (Sección 14.1). Se deben considerar dos tipos de transacciones. Las **transacciones locales** son las que acceden a los datos y los actualizan en una única base de datos local; las **transacciones globales** son las que acceden a los datos y los actualizan en varias bases de datos locales. Las propiedades ACID de las transacciones locales se pueden asegurar como se describe en los Capítulos 14, 15 y 16. No obstante, para las transacciones globales, esta tarea resulta mucho más complicada, dado que puede que participen en la ejecución varios sitios. El fallo de alguno de estos sitios, o el de alguno de los enlaces de comunicaciones que los conectan entre sí, puede dar lugar a cálculos erróneos.

En esta sección se estudia la estructura del sistema de una base de datos distribuida y sus posibles modos de fallo. Con base en el modelo presentado en esta sección, en la Sección 19.4 se estudian los protocolos para garantizar el compromiso atómico de las transacciones globales, y en la Sección 19.5 se estudian los protocolos para el control de concurrencia en las bases de datos distribuidas. En la Sección 19.6 se estudia el modo en que pueden seguir funcionando las bases de datos distribuidas incluso en presencia de varios tipos de fallos.

19.3.1. Estructura del sistema

Cada sitio tiene su propio gestor *local* de transacciones, cuya función es garantizar las propiedades ACID de las transacciones que se ejecuten en ese sitio. Los diferentes gestores de transacciones colaboran para ejecutar las transacciones globales. Para comprender cómo se pueden implementar estos gestores, considere un modelo abstracto de sistema de transacciones, en el que cada sitio contenga dos subsistemas:

- El **gestor de transacciones** gestiona la ejecución de las transacciones (o subtransacciones) que acceden a los datos almacenados en un sitio local. Tenga en cuenta que cada una de esas transacciones puede ser local (es decir, una transacción que se ejecuta solo en ese sitio) o formar parte de una transacción global (es decir, una transacción que se ejecuta en varios sitios).
- El **coordinador de transacciones** coordina la ejecución de las diferentes transacciones (tanto locales como globales) iniciadas en ese sitio.

La arquitectura global del sistema aparece en la Figura 19.2.

La estructura de los gestores de transacciones es parecida en muchos aspectos a la de los sistemas centralizados. Cada gestor de transacciones es responsable de:

- Mantener un registro histórico con fines de recuperación.
- Participar en un esquema adecuado de control de concurrencia para coordinar la ejecución concurrente de las transacciones que se ejecuten en ese sitio.

Como se verá, es necesario modificar tanto el esquema de recuperación como el de concurrencia para adaptarlos a la distribución de las transacciones.

El subsistema del coordinador de transacciones no es necesario en los entornos centralizados, ya que las transacciones solo acceden a los datos de un sitio. Los coordinadores de transacciones, como su propio nombre indica, son responsables de la coordinación de la ejecución de todas las transacciones iniciadas en ese sitio. En cada una de esas transacciones el coordinador es responsable de:

- Iniciar la ejecución de la transacción.
- Dividir la transacción en varias subtransacciones y distribuir esas subtransacciones a los sitios correspondientes para su ejecución.
- Coordinar la terminación de la transacción, lo que puede hacer que la transacción se comprometa o se aborte en todos los sitios.

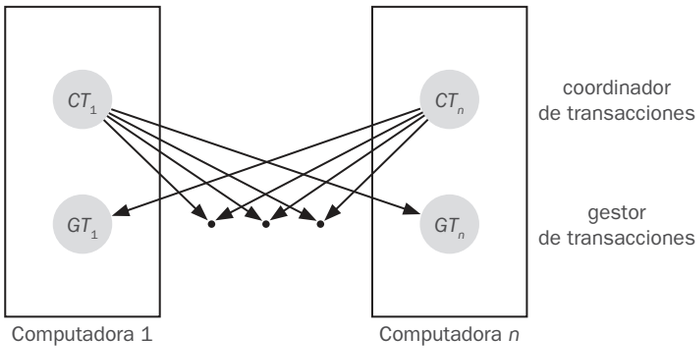


Figura 19.2. Arquitectura del sistema.

19.3.2. Modos de fallo del sistema

Los sistemas distribuidos pueden sufrir los mismos tipos de fallos que los sistemas centralizados (por ejemplo, errores de software, errores de hardware y fallos de discos). No obstante, en los entornos distribuidos también hay que tratar con otros tipos de fallos. Los tipos básicos de fallos son:

- Fallo de sitios.
- Pérdida de mensajes.
- Fallo de enlaces de comunicaciones.
- División de la red.

En los sistemas distribuidos siempre es posible la pérdida o el deterioro de los mensajes. El sistema utiliza protocolos de control de las transmisiones, como TCP/IP, para tratar esos errores. Se puede encontrar información sobre esos protocolos en los libros de texto estándar sobre redes (consulte las notas bibliográficas).

No obstante, si dos sitios A y B no se hallan conectados de manera directa, los mensajes de uno a otro deben *encaminarse* mediante una serie de enlaces de comunicaciones. Si falla uno de los enlaces, hay que volver a encaminar los mensajes que debería haber transmitido. En algunos casos se puede hallar otra ruta por la red, de modo que los mensajes puedan alcanzar su destino. En otros casos, el fallo puede hacer que no haya ninguna conexión entre los dos sitios. Un sistema está **dividido** si se ha partido en dos (o más) subsistemas, denominados **particiones,** que carecen de conexión entre ellas. Observe que, con esta definición, un subsistema puede consistir en un solo nodo.

19.4. Protocolos de compromiso

Si hay que asegurar la atomicidad, todos los sitios en los que se ejecute una transacción T deben coincidir en el resultado final de esa ejecución. T debe comprometerse o abortarse en todos los sitios. Para garantizar esta propiedad, el coordinador de transacciones de T debe ejecutar un *protocolo de compromiso*.

Entre los protocolos de compromiso más sencillos y utilizados está el **protocolo de compromiso de dos fases (C2F),** que se describe en la Sección 19.4.1. Una alternativa es el **protocolo de compromiso de tres fases (C3F),** que evita ciertos inconvenientes del protocolo C2F pero añade complejidad y sobrecarga. La Sección 19.4.2 describe brevemente el protocolo C3F.

19.4.1. Compromiso de dos fases

En primer lugar se describe el modo en que opera el protocolo de compromiso de dos fases (C2F) durante el funcionamiento normal, luego se describe cómo maneja los fallos y, finalmente, la manera en que ejecuta la recuperación y el control de concurrencia.

Considere una transacción T iniciada en el sitio S_i, en el que el coordinador de transacciones es C_i.

19.4.1.1. El protocolo de compromiso

Cuando se acaba de ejecutar T (es decir, cuando todos los sitios en los que se ha ejecutado informan a C_i de que T se ha completado) C_i inicia el protocolo C2F.

- **Fase 1.** C_i añade el registro <preparar T> al registro histórico y obliga a que se guarde en un lugar de almacenamiento estable. Luego envía el mensaje preparar T a todos los sitios en los que se ha ejecutado T. Al recibir este mensaje, el gestor de transacciones de cada sitio determina si desea comprometer su parte de T. Si la respuesta es negativa, añade el registro <no T> al registro histórico y responde enviando a C_i el mensaje abortar T. Si la respuesta es positiva, añade el registro <preparada T> al registro histórico y hace que se guarde (con todos los registros del registro histórico correspondientes a T) en almacenamiento estable. El gestor de transacciones contesta entonces a C_i con el mensaje preparada T.
- **Fase 2.** Cuando C_i recibe de todos los sitios las respuestas al mensaje preparar T, o cuando ha transcurrido un intervalo de tiempo especificado con anterioridad desde que se envió el mensaje preparar T, C_i puede determinar si la transacción T puede comprometerse o abortarse. La transacción T se puede comprometer si C_i ha recibido el mensaje preparada T de todos los sitios participantes. En caso contrario, hay que abortar la transacción T. En función del resultado, se añade el registro <T comprometida> o <T abortada> al registro histórico, que se guarda en almacenamiento estable. En ese momento, el destino de la transacción ya se ha sellado. A partir de este momento el coordinador envía a todos los sitios participantes el mensaje comprometer T o abortar T. Cuando un sitio recibe ese mensaje, lo guarda en el registro histórico.

Los sitios en los que se ejecutó T pueden abortarla de manera incondicional en cualquier momento antes de enviar al coordinador el mensaje preparada T. Una vez enviado el mensaje, se dice que la transacción está en **estado preparado** en el sitio. El mensaje preparada T constituye, en realidad, un compromiso del sitio de acatar la orden del coordinador de comprometer o de abortar T. Para establecer ese compromiso, primero hay que guardar en almacenamiento estable la información necesaria. En caso contrario, si el sitio fallara tras enviar el mensaje preparada T, puede que no fuera capaz de cumplir su promesa. Además, los bloqueos adquiridos por la transacción deben mantenerse hasta que esta se complete.

Dado que se exige la unanimidad para comprometer cada transacción, el destino de T queda sellado en cuanto un sitio responda abortar T. Dado que el sitio coordinador S_i es uno de los sitios en los que se ha ejecutado T, el coordinador puede decidir unilateralmente abortarla. El veredicto final sobre T se determina en el momento en que el coordinador lo escribe (comprometer o abortar) en el registro histórico y obliga a que ese veredicto se guarde en almacenamiento estable. En algunas implementaciones del protocolo C2F, los sitios envían al coordinador el mensaje acuse-de-recibo T al final de la segunda fase del protocolo. Cuando el coordinador recibe el mensaje acuse-de-recibo T de todos los sitios, añade el registro <T completada> al registro histórico.

19.4.1.2. Tratamiento de los fallos

El protocolo C2F responde de modo diferente a los distintos tipos de fallos:

- **Fallo de un sitio participante.** Si el coordinador C_i detecta que un sitio ha fallado, emprende las acciones siguientes: si el sitio falla antes de responder a C_i con el mensaje preparada T, el coordinador da por supuesto que ha respondido con el mensaje abortar T. Si el sitio falla después de que el coordinador haya recibido del sitio el mensaje preparada T, el coordinador ejecuta el resto del protocolo de compromiso de manera normal, ignorando el fallo del sitio.

  Cuando el sitio participante S_k se recupera de un fallo, debe examinar su registro histórico para determinar el destino de las transacciones que se hallaban en fase de ejecución cuando se produjo ese fallo. Suponga que T es una de esas transacciones. Se toman en consideración cada uno de los casos posibles:
  - El registro histórico contiene el registro <T comprometida>. En ese caso, el sitio ejecuta rehacer*(T)*.
  - El registro histórico contiene el registro <T abortada>. En ese caso, el sitio ejecuta deshacer*(T)*.

- El registro histórico contiene el registro <preparada T>. En ese caso, el sitio debe consultar con C_i para determinar el destino de T. Si C_i está activo, notifica a S_k si T se comprometió o abortó. En el primer caso, se ejecuta rehacer*(T)*; en el segundo, se ejecuta deshacer*(T)*. Si C_i no está activo, S_k debe intentar averiguar el destino de T consultando a otros sitios. Lo hace enviando el mensaje consulta-estado T a todos los sitios del sistema. Al recibir este mensaje, cada sitio debe consultar su registro histórico para determinar si en él se ejecutó T y, en caso afirmativo, si se comprometió o se abortó. Luego notifica a S_k el resultado. Si ningún sitio tiene la información correspondiente (es decir, si T se comprometió o se abortó), S_k no puede abortar ni comprometer T. La decisión sobre T se pospone hasta que S_k pueda obtener la información necesaria. Por tanto, S_k debe volver a enviar de manera periódica el mensaje consulta-estado a los demás sitios. Seguirá haciéndolo hasta que se recupere algún sitio que contenga la información necesaria. Tenga en cuenta que el sitio en el que reside C_i siempre tiene la información necesaria.
- El registro histórico no contiene ningún registro de control (abortada, comprometida, preparada) relativo a T. Por tanto, se sabe que S_k falló antes de responder al mensaje preparada T enviado por C_i. Dado que el fallo de S_k evitó el envío de la respuesta, de acuerdo con el algoritmo, C_i debe abortar T. Por tanto, S_k debe ejecutar deshacer*(T)*.

- **Fallo del coordinador.** Si el coordinador falla durante la ejecución del protocolo de compromiso para la transacción T, los sitios participantes deben decidir el destino de T. Se verá que, en ciertos casos, los sitios participantes no pueden decidir si comprometer o abortar T y, por tanto, deben esperar a la recuperación del coordinador que ha fallado.
  - Si algún sitio activo contiene el registro <T comprometida> en su registro histórico, se debe comprometer T.
  - Si algún sitio activo contiene el registro <T abortada> en su registro histórico, se debe abortar T.
  - Si algún sitio activo *no* contiene el registro <T preparada> en su registro histórico, el coordinador C_i que ha fallado no puede haber decidido comprometer T, ya que los sitios que no contienen el registro <T preparada> en su registro histórico no pueden haber enviado el mensaje T preparada a C_i. No obstante, puede que el coordinador haya decidido abortar T, pero no comprometer T. En vez de esperar a que se recupere C_i, resulta preferible abortar T.
  - Si no se da ninguno de los casos anteriores, todos los sitios activos deben tener el registro <T preparada> en sus registros históricos, pero ningún otro registro de control (como <T abortada> o <T comprometida>). Dado que el coordinador ha fallado, resulta imposible determinar si se ha tomado alguna decisión y, en caso de haberse tomado, averiguar la que era, hasta que se recupere el coordinador. Por tanto, los sitios activos deben esperar a que se recupere C_i. Dado que el destino de T sigue siendo dudoso, puede que siga consumiendo recursos del sistema. Por ejemplo, si se emplean bloqueos, puede que T conserve en los sitios activos los bloqueos sobre los datos. Esa situación no es deseable, ya que pueden pasar horas o días antes de que C_i vuelva a estar activo. Durante ese tiempo puede que otras transacciones se vean obligadas a esperar a T. En consecuencia, puede que los elementos de datos no estén disponibles, no solo en el sitio que ha fallado (C_i), sino también en los sitios activos. Esta situación se denomina problema del **bloqueo,** ya que T queda bloqueada a la espera de la recuperación del sitio C_i.

- **División de la red.** Cuando una red queda dividida, caben dos posibilidades:
  1. El coordinador y todos los sitios participantes siguen en una de las particiones. En este caso, el fallo no tiene ningún efecto sobre el protocolo de compromiso.
  2. El coordinador y los sitios participantes se distribuyen entre varias particiones. Desde el punto de vista de los sitios de cada partición, parece que los sitios de las demás particiones hubieran fallado. Los sitios que no se hallan en la partición que contiene al coordinador, sencillamente, ejecutan el protocolo para tratar el fallo del coordinador. El coordinador y los sitios que se hallan en su misma partición siguen el protocolo de compromiso habitual, dando por supuesto que los sitios de las demás particiones han fallado.

Por tanto, el mayor inconveniente del protocolo C2F es que el fallo del coordinador puede dar lugar a un bloqueo, en el que puede que haya que retrasar la decisión sobre comprometer o abortar T hasta que se recupere C_i.

19.4.1.3. Recuperación y control de concurrencia

Cuando se reinicia el sitio que ha fallado, la recuperación se puede llevar a cabo, por ejemplo, utilizando el algoritmo de recuperación descrito en la Sección 16.4. Para tratar con los protocolos de compromiso distribuidos, el procedimiento de recuperación debe tratar de manera especial las **transacciones dudosas;** las transacciones dudosas son transacciones para las que se encuentra en el registro histórico un registro <preparada T>, pero no <T comprometida> ni <T abortada>. El sitio que se recupera debe determinar la situación comprometer-abortar de esas transacciones, como se describe en la Sección 19.4.1.2.

Sin embargo, si se realiza la recuperación como se acaba de describir, el procesamiento normal de las transacciones en el sitio no puede comenzar hasta que se hayan comprometido o deshecho todas las transacciones dudosas. Averiguar la situación de las transacciones dudosas puede ser un proceso lento, ya que es posible que haya que contactar con varios sitios. Además, si ha fallado el coordinador, y ningún otro sitio tiene información sobre la situación comprometer-abortar de una transacción incompleta, se podría bloquear la recuperación si se utiliza C2F. En consecuencia, puede que el sitio que lleva a cabo la recuperación de reinicio quede inutilizable durante un largo periodo de tiempo.

Para evitar este problema, los algoritmos de recuperación suelen ofrecer soporte para anotar en el registro histórico la información relativa a los bloqueos (se está suponiendo que se utilizan los bloqueos para el control de concurrencia). En lugar de escribir en el registro histórico el registro <preparada T>, el algoritmo escribe <preparada T, L>, donde L es una lista de todos los bloqueos de escritura que tiene la transacción T cuando se escribe el registro del registro histórico. En el momento de la recuperación, tras llevar a cabo las acciones de recuperación locales, se renuevan para cada transacción dudosa T todos los bloqueos de escritura anotados en el registro <preparada T, L> del registro histórico (tras leerlos en el registro histórico).

Después de que se haya completado la renovación de los bloqueos para todas las transacciones dudosas, puede comenzar en el sitio el procesamiento de las transacciones, incluso antes de que se determine el estado comprometer-abortar de las transacciones dudosas. Las transacciones dudosas se comprometen o se deshacen de manera concurrente con la ejecución de las nuevas transacciones. Así, la recuperación del sitio es más rápida y no se bloquea nunca. Observe que las transacciones nuevas que tengan conflictos de bloqueo con algún bloqueo de escritura establecido por cualquier transacción dudosa no pueden progresar hasta que se hayan comprometido o deshecho las transacciones dudosas con las que estén en conflicto.

19.4.2. Compromiso de tres fases

El protocolo de compromiso de tres fases (C3F) es una extensión del protocolo de compromiso de dos fases que evita el problema del bloqueo bajo determinados supuestos. En concreto, se supone que no se produce división de la red y que no fallan más de k sitios, donde k es un número predeterminado. Con estos supuestos, el protocolo evita el bloqueo introduciendo una tercera fase adicional en la que varios sitios se implican en la decisión sobre el compromiso. En lugar de anotar directamente la decisión sobre el compromiso en su almacenamiento persistente, el coordinador se asegura antes de que, al menos, otros k sitios sepan que pretende comprometer la transacción. Si el coordinador falla, los sitios restantes seleccionan primero un nuevo coordinador. Este nuevo coordinador comprueba el estado del protocolo a partir de los demás sitios; si el coordinador había decidido comprometer, al menos uno de los otros k sitios a los que informó estará funcionando y garantizará que se respeta la decisión de comprometer. El nuevo coordinador vuelve a iniciar la tercera fase del protocolo si algún sitio sabía que el antiguo coordinador pretendía comprometer la transacción. En caso contrario, el nuevo coordinador la aborta.

Aunque el protocolo C3F tiene la propiedad deseable de no bloquearse a menos que fallen k sitios, tiene el inconveniente de que una división de la red puede parecer lo mismo que el fallo de más de k sitios, lo que produce un bloqueo. El protocolo también tiene que implementarse con mucho cuidado para garantizar que la división de la red (o el fallo de más de k sitios) no provoque inconsistencias, en las que una transacción se comprometa en una de las particiones y se aborte en otra. Debido a la sobrecarga que supone, el protocolo C3F no se utiliza mucho. Consulte las notas bibliográficas para más referencias sobre el protocolo C3F.

19.4.3. Modelos alternativos del procesamiento de transacciones

Para muchas aplicaciones, el problema del bloqueo del compromiso de dos fases no resulta aceptable. El problema en este caso es la idea de una sola transacción que trabaja en varios sitios. En esta sección se describe el modo de utilizar la *mensajería persistente* para evitar el problema del compromiso distribuido y luego se describe brevemente el problema, más importante, de los *flujos de trabajo*; los flujos de trabajo se tratan con mayor detalle en la Sección 26.2.

Para comprender la mensajería persistente, considere el modo en que se podrían transferir fondos entre dos bancos diferentes, cada uno con su propia computadora. Un enfoque consiste en hacer que la transacción incluya los dos sitios y utilizar el compromiso de dos fases para asegurar la atomicidad. Sin embargo, puede que la transacción tenga que actualizar todo el saldo del banco, y el bloqueo podría afectar gravemente a las demás transacciones de cada banco, ya que casi todas las transacciones del banco actualizan el saldo total de este.

Por el contrario, considere el modo en que se produce la transferencia de fondos mediante cheque conformado. El banco deduce en primer lugar el importe del cheque del saldo disponible e imprime un cheque. El cheque se transfiere físicamente al otro banco, donde se ingresa. Tras comprobar el cheque, el banco incrementa el saldo local en el importe del cheque. El cheque constituye un mensaje enviado entre los dos bancos. Para que los fondos no se pierdan ni se incrementen de manera incorrecta, el cheque no se debe perder, duplicar ni ingresar más de una vez. Cuando las computadoras de los bancos se hallan conectadas en red, los mensajes persistentes ofrecen el mismo servicio que los cheques (pero, por supuesto, mucho más rápido).

Los **mensajes persistentes** son mensajes que tienen garantizada su entrega al destinatario exactamente una sola vez (ni más, ni menos), independientemente de los fallos, si la transacción que envía el mensaje se compromete, y tienen garantizado que no se entregan si la transacción se aborta. Se emplean técnicas de recuperación de las bases de datos para implementar la mensajería persistente por encima de los canales normales de la red, como se verá brevemente. Por el contrario, los mensajes normales se pueden perder o, incluso, se pueden entregar varias veces en determinadas circunstancias.

El manejo de los errores resulta más complicado con la mensajería persistente que con el compromiso de dos fases. Por ejemplo, si la cuenta en la que hay que ingresar el cheque se ha cerrado, hay que devolver este a la cuenta que lo originó y volver a cargar su importe en ella. Por tanto, ambos sitios deben disponer de código para el manejo de errores y código para manejar los mensajes persistentes. Por el contrario, con el compromiso de dos fases, el error lo detectaría la propia transacción, que, por tanto, no deduciría nunca el importe del cheque de la primera cuenta.

Los tipos de condiciones de excepción que pueden surgir dependen de la aplicación, por lo que no es posible que el sistema de bases de datos maneje las excepciones de manera automática. Los programas de aplicación que envían y reciben los mensajes persistentes deben incluir código para el manejo de las condiciones de excepción y para que el sistema vuelva a un estado consistente. Por ejemplo, no resulta aceptable que se pierda el dinero que se iba a transferir porque la cuenta receptora se haya cerrado; hay que devolver el dinero a la cuenta ordenante, y si ello no resulta posible por algún motivo, hay que advertir a los empleados del banco para que resuelvan manualmente el problema.

Existen muchas aplicaciones en las que la ventaja de la eliminación del bloqueo compensa ampliamente el esfuerzo adicional de implementar sistemas que utilicen mensajes persistentes. De hecho, pocas organizaciones aceptarían soportar el compromiso de dos fases en transacciones que se originen en su exterior, ya que los fallos pueden provocar el bloqueo del acceso a los datos locales. La mensajería persistente, por tanto, desempeña un papel importante en la ejecución de las transacciones que cruzan las fronteras de las organizaciones.

Los *flujos de trabajo* proporcionan un modelo general de procesamiento de las transacciones que implican a varios sitios y, posiblemente, el procesamiento manual por los empleados de la organización de determinadas fases del proceso. Por ejemplo, cuando un banco recibe una solicitud de préstamo, debe dar muchos pasos, incluido el contacto con agencias externas de calificación de crédito, antes de aceptar o rechazar esa solicitud. Esos pasos, en su conjunto, forman un flujo de trabajo. Los flujos de trabajo se estudian con mayor detalle en la Sección 26.2. También se observa que la mensajería persistente está en la base de los flujos de trabajo en entornos distribuidos.

Se considerará ahora la **implementación** de la mensajería persistente. Se puede implementar la mensajería persistente utilizando una infraestructura de mensajería que no sea fiable, es decir, que pueda perder mensajes o entregarlos varias veces; para ello se pueden seguir los siguientes protocolos:

- **Protocolo del sitio remitente.** Cuando una transacción desea enviar un mensaje persistente, escribe el registro que contiene el mensaje en la relación especial *mensajes_por_enviar*, en lugar de enviar el mensaje directamente. El mensaje también recibe un identificador de mensaje único.

  Un *proceso de entrega de mensajes* monitoriza la relación y, cuando detecta un mensaje nuevo, lo envía a su destino. Los mecanismos habituales de control de concurrencia de las bases de datos garantizan que el proceso del sistema solo lea el mensaje una vez que la transacción que lo ha escrito se haya comprometido; si la transacción se aborta, el mecanismo habitual de recuperación borra el mensaje de la relación.

El proceso de entrega de mensajes solo elimina los mensajes de la relación una vez que ha recibido un acuse de recibo del sitio de destino. Si no lo recibe, pasado cierto tiempo vuelve a enviar el mensaje. Repite este proceso hasta que recibe un acuse de recibo. En caso de fallo permanente, el sistema decide, pasado algún tiempo, que el mensaje no puede entregarse. Entonces invoca al código proporcionado por la aplicación para el manejo de excepciones para que trate el fallo.

La escritura del mensaje en una relación y su procesamiento tan solo después de que se haya comprometido la transacción garantiza que el mensaje se entregue si, y solo si, la transacción se compromete. Su envío repetido garantiza que se entregue aunque haya fallos (temporales) del sistema o de la red.

- **Protocolo del sitio receptor.** Cuando un sitio recibe un mensaje persistente, ejecuta una transacción que añade el mensaje a la relación especial *mensajes_recibidos*, siempre que no se encuentre ya presente en la relación (el identificador único de mensajes permite detectar los duplicados). Una vez comprometida la transacción, o si el mensaje ya se hallaba en la relación, el sitio receptor devuelve un acuse de recibo al sitio remitente.

  Tenga en cuenta que no resulta seguro enviar el acuse de recibo antes de que la transacción se comprometa, ya que un fallo del sistema podría dar lugar a la pérdida del mensaje. Comprobar si el mensaje se ha recibido previamente resulta fundamental para evitar que se entregue varias veces.

  En muchos sistemas de mensajería, los mensajes se pueden retrasar de manera arbitraria, aunque esos retrasos sean muy improbables. Por tanto, por seguridad, los mensajes no se deben eliminar nunca de la relación *mensajes_recibidos*. Su eliminación puede hacer que no se detecte una entrega duplicada. Pero, como consecuencia, la relación *mensajes_recibidos* puede crecer de manera indefinida. Para resolver este problema se asigna a cada mensaje una marca temporal y, si la marca temporal del mensaje recibido es más antigua que la de algún punto arbitrario de corte, ese mensaje se descarta. Todos los mensajes registrados en la relación *mensajes_recibidos* que sean más antiguos que el punto de corte se pueden eliminar.

19.5. Control de la concurrencia en las bases de datos distribuidas

En esta sección se muestra el modo en que se pueden modificar algunos de los esquemas de control de concurrencia que se estudiaron en el Capítulo 15 para utilizarlos en entornos distribuidos. Se supone que cada sitio participa en la ejecución de un protocolo de compromiso para garantizar la atomicidad global de las transacciones.

Los protocolos que se describen en esta sección necesitan que se hagan actualizaciones de todas las réplicas de los elementos de datos. Si ha fallado algún sitio que contenga una réplica de un elemento de datos, no se pueden procesar las actualizaciones de ese elemento de datos. En la Sección 19.6 se describen los protocolos que pueden continuar el procesamiento de las transacciones aunque haya fallado algún sitio o algún enlace, lo que proporciona una gran disponibilidad.

19.5.1. Protocolos de bloqueo

Se pueden utilizar los distintos protocolos de bloqueo descritos en el Capítulo 15 en entornos distribuidos. La única modificación que hay que incorporar es el modo en que el gestor de bloqueos trata los datos replicados. Se presentan varios esquemas posibles que son aplicables a entornos en los que los datos se pueden replicar en varios sitios. Al igual que en el Capítulo 15, se supondrá la existencia de los modos de bloqueo *compartido* y *exclusivo*.

19.5.1.1. Enfoque de gestor único de bloqueos

En el enfoque de **gestor único de bloqueos,** el sistema mantiene un *único* gestor de bloqueos que reside en un sitio *único* escogido (por ejemplo, S_i). Todas las solicitudes de bloqueo y de desbloqueo se realizan en el sitio S_i. Cuando una transacción necesita bloquear un elemento de datos, envía una solicitud de bloqueo a S_i. El gestor de bloqueos determina si se puede conceder el bloqueo de manera inmediata. En caso afirmativo, envía un mensaje al respecto al sitio en el que se inició la solicitud de bloqueo. En caso contrario, la solicitud se retrasa hasta que se puede conceder, en cuyo momento se envía un mensaje al sitio en el que se inició la solicitud de bloqueo. La transacción puede leer el elemento de datos de *cualquiera* de los sitios en los que residan sus réplicas. En el caso de las operaciones de escritura, deben implicarse todos los sitios en los que residan réplicas del elemento de datos.

El esquema tiene las ventajas siguientes:

- **Implementación sencilla.** Este esquema necesita dos mensajes para tratar las solicitudes de bloqueo y solo uno para las de desbloqueo.
- **Tratamiento sencillo de los interbloqueos.** Como todas las solicitudes de bloqueo y de desbloqueo se realizan en un solo sitio, se pueden aplicar directamente a este entorno los algoritmos de tratamiento de los interbloqueos estudiados en el Capítulo 15.

Los inconvenientes del esquema son:

- **Cuello de botella.** El sitio S_i se transforma en un cuello de botella, ya que todas las solicitudes deben procesarse allí.
- **Vulnerabilidad.** Si el sitio S_i falla, se pierde el controlador de la concurrencia. O bien hay que detener el procesamiento, o bien hay que utilizar un esquema de recuperación para que un sitio de respaldo pueda asumir la administración de los bloqueos de S_i, como se describe en la Sección 19.6.5.

19.5.1.2. Gestor distribuido de bloqueos

Se puede lograr un compromiso entre las ventajas y los inconvenientes ya mencionados mediante el enfoque del **gestor distribuido de bloqueos,** en el que la función de gestor de bloqueos se distribuye entre varios sitios.

Cada sitio mantiene un gestor de bloqueos local, cuya función es administrar las solicitudes de bloqueo y de desbloqueo para los elementos de datos que se almacenan en ese sitio. Cuando una transacción desea bloquear el elemento de datos Q, que no está replicado y reside en el sitio S_i, envía un mensaje al gestor de bloqueos del sitio S_i para solicitarle un bloqueo (en un modo de bloqueo determinado). Si el elemento de datos Q está bloqueado en un modo incompatible, la solicitud se retrasa hasta que se pueda conceder. Una vez se haya determinado que la solicitud de bloqueo se puede conceder, el gestor de bloqueos devuelve un mensaje al sitio que ha iniciado la solicitud para indicar que ha concedido la solicitud de bloqueo.

Existen varios modos alternativos de tratar con la réplica de los elementos de datos, que se estudian en las Secciones 19.5.1.3 a 19.5.1.6.

El esquema del gestor distribuido de bloqueos presenta la ventaja de su sencilla implementación y de que reduce el grado en el que el coordinador constituye un cuello de botella. Tiene una sobrecarga razonablemente baja, ya que solo necesita dos transferencias de mensajes para tratar las solicitudes de bloqueo y una transferencia de mensaje para las de desbloqueo. Sin embargo, el tratamiento de los interbloqueos resulta más complejo, dado que las solicitudes de bloqueo y de desbloqueo ya no se realizan en un solo sitio. Puede haber interbloqueos entre sitios aunque no los haya dentro de ninguno de los sitios. Hay que modificar los algoritmos de tratamiento de los interbloqueos estudiados en el Capítulo 15, como se estudiará en la Sección 19.5.4, para detectar interbloqueos globales.

19.5.1.3. Copia principal

Cuando un sistema utiliza replicación de datos se puede escoger una de las réplicas como **copia principal.** Así, para cada elemento de datos Q, la copia principal de Q debe residir exactamente en un sitio, que se denomina **sitio principal** de Q.

Cuando una transacción necesita bloquear el elemento de datos Q, solicita un bloqueo en el sitio principal de Q. Como ya se ha visto, la respuesta a la solicitud se retrasa hasta que pueda concederse. Por tanto, la copia principal permite que el control de concurrencia de los datos replicados se trate como el de los datos no replicados. Esta semejanza permite una implementación sencilla. No obstante, si falla el sitio principal de Q, ese elemento de datos queda inaccesible, aunque otros sitios que contengan réplicas suyas estén accesibles.

19.5.1.4. Protocolo de mayoría

El **protocolo de mayoría** funciona de la siguiente manera: si el elemento de datos Q se replica en n sitios diferentes, hay que enviar un mensaje de solicitud de bloqueo a más de la mitad de esos sitios. Cada gestor de bloqueos determina si se puede conceder el bloqueo de manera inmediata (en lo que a él se refiere). Como anteriormente, la respuesta se retrasa hasta que la solicitud se pueda conceder. La transacción no se lleva a cabo en Q hasta que logre obtener un bloqueo sobre la mayoría de las réplicas de Q.

Se va a suponer por ahora que las operaciones de escritura se llevan a cabo en todas las réplicas, lo que exige que todos los sitios que las contienen estén disponibles. No obstante, la principal ventaja del protocolo de mayoría es que puede extenderse para que trate los fallos de los sitios, como se verá en la Sección 19.6.1. Este protocolo también trata los datos replicados de manera descentralizada, con lo que evita los inconvenientes del control centralizado. Sin embargo, presenta los siguientes inconvenientes:

- **Implementación.** El protocolo de mayoría es más complicado de implementar que los esquemas anteriores. Necesita, como mínimo, $2(n/2 + 1)$ mensajes para manejar las solicitudes de bloqueo y, al menos, $(n/2 + 1)$ mensajes para manejar las solicitudes de desbloqueo.
- **Tratamiento de los interbloqueos.** Además del problema de los interbloqueos globales debidos al empleo del enfoque del gestor distribuido de bloqueos, puede que se produzcan interbloqueos aunque solo se esté bloqueando un elemento de datos. A modo de ejemplo, considere un sistema con cuatro sitios y réplica completa. Suponga que las transacciones T_1 y T_2 desean bloquear el elemento de datos Q en modo exclusivo. Puede que la transacción T_1 tenga éxito en el bloqueo de Q en los sitios S_1 y S_3 y la transacción T_2 consiga bloquear Q en los sitios S_2 y S_4. Cada una de ellas deberá esperar a adquirir el tercer bloqueo; por tanto, se ha producido un interbloqueo. Por fortuna, estos interbloqueos se pueden evitar con relativa facilidad exigiendo que todos los sitios soliciten el bloqueo de las réplicas de cada elemento de datos en el mismo orden predeterminado.

19.5.1.5. Protocolo sesgado

El **protocolo sesgado** es otro enfoque del manejo de las réplicas. La diferencia con el protocolo de mayoría es que se concede un tratamiento más favorable a las solicitudes de bloqueo compartido que a las solicitudes de bloqueo exclusivo.

- **Bloqueos compartidos.** Cuando una transacción necesita bloquear el elemento de datos Q, simplemente solicita su bloqueo al gestor de bloqueos de un sitio que contenga una réplica de Q.
- **Bloqueos exclusivos.** Cuando una transacción necesita bloquear el elemento de datos Q, solicita su bloqueo al gestor de bloqueos de todos los sitios que contienen una réplica de Q.

Al igual que antes, la respuesta a la solicitud se retrasa hasta que pueda concederse.

El esquema sesgado tiene la ventaja de imponer menos sobrecarga a las operaciones leer que el protocolo de mayoría. Este ahorro resulta especialmente significativo en el frecuente caso en que la frecuencia de las operaciones leer sea mucho mayor que la de las operaciones escribir. No obstante, la sobrecarga adicional sobre las operaciones de escritura supone un inconveniente. Además, el protocolo sesgado comparte con el protocolo de mayoría el inconveniente de la complejidad en el manejo de los interbloqueos.

19.5.1.6. Protocolo de consenso de quórum

El protocolo de **consenso de quórum** es una generalización del protocolo de mayoría. El protocolo de consenso de quórum asigna a cada sitio un peso no negativo. Asigna a las operaciones de lectura y de escritura sobre el elemento x dos enteros, denominados **quórum de lectura** Q_l y **quórum de escritura** Q_e, que deben cumplir la siguiente condición, donde S es el peso total de todos los sitios en los que x reside:

$$Q_l + Q_e > S \textbf{ y } 2 \times Q_e > S$$

Para ejecutar una operación de lectura deben bloquearse suficientes réplicas como para que su peso total sea al menos de r. Para ejecutar una operación de escritura se deben bloquear suficientes réplicas como para que su peso total sea w.

Una ventaja del enfoque de consenso de quórum es que puede permitir la reducción selectiva del coste de los bloqueos de lectura o de escritura mediante la adecuada definición de los quórum de lectura y de escritura. Por ejemplo, con un quórum de lectura pequeño, las operaciones de lectura necesitan obtener menos bloqueos, pero el quórum de escritura será mayor, por lo que las operaciones de escritura necesitarán obtener más bloqueos. Además, si se asignan pesos más elevados a algunos sitios (por ejemplo, a los que tengan menos posibilidad de fallar), hace falta acceder a menos sitios para adquirir los bloqueos. De hecho, si se definen los pesos y los quórum de manera adecuada, el protocolo de consenso de quórum puede simular tanto el protocolo de mayoría como el sesgado.

Al igual que el protocolo de mayoría, el de consenso de quórum se puede extender para que trabaje incluso en caso de fallos de los sitios, como se verá en la Sección 19.6.1.

19.5.2. Marcas temporales

La idea principal tras el esquema de marcas temporales de la Sección 15.4 es que se concede a cada transacción una marca temporal *única* que el sistema utiliza para decidir el orden de secuenciación. La primera tarea, por tanto, al generalizar el esquema centralizado a uno distribuido es desarrollar un esquema para la generación de marcas temporales únicas. Por tanto, los diferentes protocolos pueden operar directamente en el entorno no replicado.

Existen dos métodos principales para la generación de marcas temporales únicas, uno centralizado y otro distribuido. En el esquema centralizado un solo sitio distribuye las marcas temporales. El sitio puede utilizar para ello un contador lógico o su propio reloj local.

En el esquema distribuido, cada sitio genera una marca temporal local única mediante un contador lógico o su reloj local. La marca temporal global única se obtiene concatenando la marca temporal local única correspondiente con el identificador de ese sitio, que también debe ser único (Figura 19.3). El orden de concatenación es importante. El identificador del sitio se utiliza en la posición menos significativa para garantizar que las marcas temporales globales generadas en un sitio dado no sean siempre mayores que las generadas en otro. Compare esta técnica para la generación de marcas temporales únicas con la presentada en la Sección 19.2.3 para la generación de nombres únicos.

Puede que todavía surja algún problema si un sitio genera marcas temporales locales a una velocidad mayor que los demás sitios. En ese caso, el contador lógico del sitio rápido será mayor que el de los demás sitios. Por tanto, todas las marcas temporales generadas por el sitio rápido serán mayores que las generadas por los demás sitios. Lo que se necesita es un mecanismo que garantice que las marcas temporales locales se generen de manera homogénea en todo el sistema. En cada sitio S_i se define un **reloj lógico** (RL_i), que genera la marca temporal local única. El reloj lógico puede implementarse como un contador que se incremente después de generar cada nueva marca temporal local. Para garantizar que los diferentes relojes lógicos estén sincronizados, se exige que el sitio S_i adelante su reloj lógico siempre que una transacción T_i con la marca temporal $<x,y>$ visite ese sitio y x sea mayor que el valor actual de RL_i. En ese caso, el sitio S_i adelantará su reloj lógico hasta el valor $x + 1$.

Si se utiliza el reloj del sistema para generar las marcas temporales, estas se asignarán de manera homogénea, siempre que ningún sitio tenga un reloj del sistema que adelante o atrase. Dado que es posible que los relojes no sean totalmente exactos, hay que utilizar una técnica parecida a la de los relojes lógicos para garantizar que ningún reloj se adelante o se atrase mucho respecto de los demás.

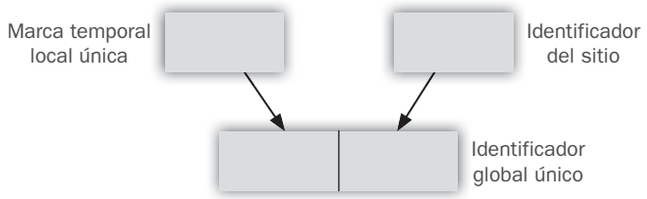


Figura 19.3. Generación de marcas temporales únicas.

19.5.3. Réplica con grado de consistencia bajo

Muchas bases de datos comerciales actuales soportan las réplicas, que pueden adoptar varias formas. Con la **réplica maestro-esclavo,** la base de datos permite las actualizaciones en el sitio principal y las propaga de manera automática a las réplicas de los demás sitios. Las transacciones pueden leer las réplicas en los demás sitios, pero no se les permite actualizarlas.

Una característica importante de esta réplica es que las transacciones no consiguen bloqueos en los sitios remotos. Para garantizar que las transacciones que se ejecutan en los sitios de réplica vean una vista consistente (aunque quizás desactualizada) de la base de datos, la réplica debe reflejar una **instantánea consistente para las transacciones** de los datos del sitio principal; es decir, la réplica debe reflejar todas las actualizaciones de las transacciones hasta una transacción dada según el orden de secuenciación, y no deben reflejar ninguna actualización de transacciones posteriores según el orden de secuenciación.

Se puede configurar la base de datos para que propague las actualizaciones de manera inmediata, una vez producidas en el sitio principal, o para que las propague solo de manera periódica.

La réplica maestro-esclavo resulta especialmente útil para distribuir información, por ejemplo, desde una oficina central a las sucursales de una organización. Otra aplicación de esta forma de réplica es la creación de copias de la base de datos para la ejecución de consultas de gran tamaño, de modo que las consultas no interfieran con las transacciones. Las actualizaciones deben propagarse de manera periódica (cada noche, por ejemplo), de modo que la propagación no interfiera con el procesamiento de las consultas.

El sistema de bases de datos de Oracle posee la sentencia **create snapshot** (crear instantánea), que puede crear en un sitio remoto una copia de una instantánea de una relación, o de un conjunto de relaciones, consistente para las transacciones. También permite la actualización de las instantáneas, que puede hacerse volviendo a calcular cada instantánea o actualizándola de manera incremental. Oracle dispone de actualización automática, tanto continua como a intervalos periódicos.

Con la **réplica multimaestro** (también denominada **réplica de actualización distribuida**) las actualizaciones se permiten en cualquier réplica de un elemento de datos y se propagan de manera automática a todas las réplicas. Este modelo es el modelo básico utilizado para administrar las réplicas en las bases de datos distribuidas. Las transacciones actualizan la copia local y el sistema actualiza las demás réplicas de manera transparente.

Un modo de actualizar las réplicas es aplicar la actualización inmediata con compromiso de dos fases, utilizando una de las técnicas de control de concurrencia distribuida que se han visto. Muchos sistemas de bases de datos utilizan el protocolo sesgado como técnica de control de concurrencia, en el que las operaciones de escritura tienen que bloquear y actualizar todas las réplicas y las operaciones de lectura bloquean y leen cualquier réplica.

Muchos sistemas de bases de datos ofrecen una forma alternativa de actualización: actualizan en un sitio dado, con **propagación perezosa** de las actualizaciones a los demás sitios, en lugar de aplicar de manera inmediata las actualizaciones a todas las réplicas como parte de la transacción que lleva a cabo la actualización. Los esquemas basados en la propagación perezosa permiten que continúe el procesamiento de las transacciones (incluidas las actualizaciones) aunque algún sitio quede desconectado de la red, lo que mejora la disponibilidad, pero, por desgracia, lo hacen a costa de la consistencia. Cuando se emplea la propagación perezosa se suele seguir uno de estos dos enfoques:

- Las actualizaciones de las réplicas se traducen en actualizaciones del sitio principal, que luego se propagan de manera perezosa a todas las réplicas. Este enfoque garantiza que las actualizaciones de cada elemento se ordenen de manera secuencial, aunque puedan producirse problemas de secuenciabilidad, ya que puede que las transacciones lean un valor antiguo de algún otro elemento de datos y lo utilicen para llevar a cabo una actualización.
- Las actualizaciones se llevan a cabo en cualquier réplica y se propagan a todas las demás. Este enfoque puede provocar todavía más problemas, ya que el mismo elemento de datos puede ser actualizado de manera concurrente en varios sitios.

Se pueden detectar algunos conflictos por la falta de control de concurrencia distribuida cuando las actualizaciones se propagan a otros sitios (se verá el modo de hacerlo en la Sección 25.5.4), pero la resolución del conflicto implica hacer retroceder transacciones comprometidas y, por tanto, no se garantiza la durabilidad de las transacciones comprometidas. Además, puede que se necesite la intervención del personal encargado para que resuelva el conflicto. Por tanto, los esquemas mencionados deben evitarse o utilizarse con precaución.

19.5.4. Tratamiento de los interbloqueos

Los algoritmos de prevención y de detección de interbloqueos del Capítulo 15 pueden utilizarse en los sistemas distribuidos, siempre que se realicen modificaciones. Por ejemplo, se puede utilizar el protocolo de árbol si se define un árbol *global* entre los elementos de datos del sistema. De manera parecida, el enfoque de ordenación por marcas temporales puede aplicarse de manera directa en entornos distribuidos, como se vio en la Sección 19.5.2.

La prevención de interbloqueos puede dar lugar a esperas y retrocesos innecesarios. Además, puede que algunas técnicas de prevención de interbloqueos necesiten que se impliquen en la ejecución de cada transacción más sitios de los que serían necesarios de otro modo.

Si se permite que los interbloqueos se produzcan y se confía en su detección, el problema principal en los sistemas distribuidos es decidir el modo en que se mantiene el grafo de espera. Las técnicas habituales para tratar este problema exigen que cada sitio guarde un **grafo local de espera.** Los nodos del grafo en un momento dado se corresponden con todas las transacciones (locales y no locales) que en ese momento tienen o solicitan alguno de los elementos locales de ese sitio. Por ejemplo, la Figura 19.4 muestra un sistema que consta de dos sitios, cada uno de los cuales mantiene su propio grafo local de espera. Observe que las transacciones T_2 y T_3 aparecen en los dos grafos, lo que indica que han solicitado elementos en los dos sitios.

Estos grafos locales de espera se crean de la manera habitual para las transacciones y los elementos de datos locales. Cuando una transacción T_i del sitio S_1 necesita un recurso del sitio S_2, envía un mensaje de solicitud al sitio S_2. Si el recurso lo tiene la transacción T_j, el sistema introduce el arco $T_i \rightarrow T_j$ en el grafo de espera local del sitio S_2.

Evidentemente, si algún grafo de espera local tiene un ciclo, se ha producido un interbloqueo. Por otro lado, el hecho de que no haya ciclos en ninguno de los grafos locales de espera no significa que no haya interbloqueos. Para ilustrar este problema, considere los grafos locales de espera de la Figura 19.4. Cada grafo de espera es acíclico y, sin embargo, hay un interbloqueo en el sistema debido a que la *unión* de los grafos locales de espera contiene un ciclo. Este grafo aparece en la Figura 19.5.

En el enfoque de **detección centralizada de interbloqueos,** el sistema crea y mantiene un **grafo global de espera** (la unión de todos los grafos locales) en un *solo* sitio: el coordinador de detección de interbloqueos. Dado que hay un retraso en las comunicaciones en el sistema, hay que distinguir entre dos tipos de grafos de espera. Los grafos *reales* describen el estado real pero desconocido del sistema en un momento dado, como lo vería un observador omnisciente. Los grafos *creados* son una aproximación generada por el controlador durante la ejecución de su algoritmo. Evidentemente, el controlador debe generar el grafo creado de modo que, siempre que se invoque al algoritmo de detección, los resultados obtenidos sean correctos. *Correcto* significa en este caso que, si hay algún interbloqueo, se comunique con prontitud y, si el sistema comunica algún interbloqueo, realmente se halle en estado de interbloqueo.

El grafo global de espera debe poder volver a crearse o a actualizarse bajo las siguientes condiciones:

- Siempre que se introduzca o se elimine un nuevo arco en alguno de los grafos locales de espera.
- De manera periódica, cuando se hayan producido varias modificaciones en los grafos locales de espera.
- Siempre que el coordinador necesite invocar el algoritmo de detección de ciclos.

Cuando el coordinador invoca el algoritmo de detección de interbloqueos, busca en su grafo global. Si descubre un ciclo, selecciona una víctima para hacer que retroceda. El coordinador debe comunicar a todos los sitios que se ha seleccionado como víctima a una determinada transacción. Los sitios, a su vez, hacen retroceder a dicha transacción víctima.

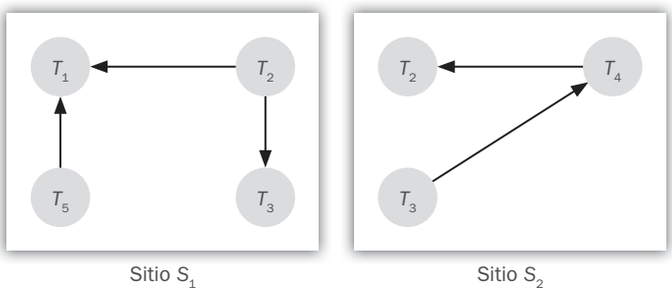


Figura 19.4. Grafos locales de espera.

Este esquema puede producir retrocesos innecesarios si:

- Existen **ciclos falsos** en el grafo global de espera. A modo de ejemplo, considere una instantánea del sistema representado por los grafos locales de espera de la Figura 19.6. Suponga que T_2 libera el recurso que tiene en el sitio S_1, lo que provoca la eliminación del arco $T_1 \rightarrow T_2$ de S_1. La transacción T_2 solicita entonces un recurso que tiene T_3 en el sitio S_2, lo que da lugar a que se añada el arco $T_2 \rightarrow T_3$ en S_2. Si el mensaje de S_2 insertar $T_2 \rightarrow T_3$ llega antes que el mensaje eliminar $T_1 \rightarrow T_2$ de S_1, puede que el coordinador descubra el ciclo falso $T_1 \rightarrow T_2 \rightarrow T_3$ después del mensaje insertar (pero antes del mensaje eliminar). Puede que se inicie la recuperación de interbloqueos, aunque no se haya producido ninguno.

  Observe que la situación de ciclos falsos no se puede producir con bloqueos de dos fases. La probabilidad de aparición de ciclos falsos suele ser lo bastante baja como para que no generen un problema serio de rendimiento.
- Se produce realmente un *interbloqueo* y se escoge una víctima cuando resulta que se ha abortado alguna de las transacciones por motivos no relacionados con el interbloqueo. Por ejemplo, suponga que el sitio S_1 de la Figura 19.4 decide abortar T_2. Al mismo tiempo, el coordinador ha descubierto un ciclo y ha escogido como víctima a T_3. Se hace que retrocedan tanto T_2 como T_3, aunque solo sería necesario hacer que retrocediera T_2.

La detección de interbloqueos puede hacerse de manera distribuida, con varios sitios que asuman partes de la tarea, en lugar de hacerla en un solo sitio. No obstante, los algoritmos correspondientes resultan más complicados y costosos. Consulte las notas bibliográficas para más información sobre estos algoritmos.

19.6. Disponibilidad

Uno de los objetivos del empleo de bases de datos distribuidas es disponer de una **disponibilidad elevada;** es decir, la base de datos debe funcionar casi todo el tiempo. En concreto, dado que los fallos son más probables en los sistemas distribuidos de gran tamaño, las bases de datos distribuidas deben seguir funcionando aunque sufran diferentes tipos de fallos. La posibilidad de continuar funcionando incluso durante los fallos se denomina **robustez.**

Para que un sistema distribuido sea robusto debe *detectar* los fallos, *reconfigurar* el sistema de modo que el cálculo pueda continuar y *recuperarse* cuando se repare el procesador o el enlace.

Los diferentes tipos de fallos se manejan de manera diferente. Por ejemplo, la pérdida de mensajes se trata mediante su retransmisión. La retransmisión repetida de un mensaje por un enlace, sin la recepción de un acuse de recibo, suele ser síntoma de un fallo de ese enlace. La red intentará hallar una ruta alternativa para el mensaje. La imposibilidad de hallar esa ruta suele ser síntoma de una división de la red.

No obstante, no suele ser posible diferenciar claramente entre los fallos de los sitios y las divisiones de la red. El sistema, generalmente, puede detectar que se ha producido un fallo, pero puede que no logre identificar su tipo. Por ejemplo, suponga que el sitio S_1 no puede comunicar con S_2. Puede ser que S_2 haya fallado. No obstante, otra posibilidad es que el enlace entre S_1 y S_2 haya fallado, lo que habrá provocado la división de la red. El problema se aborda en parte empleando varios enlaces entre los diferentes sitios, de modo que, aunque falle algún enlace, sigan conectados. Sin embargo, todavía pueden fallar varios enlaces simultáneamente, por lo que hay situaciones en las que no se puede estar seguro de si se ha producido un fallo del sitio o una división de la red.

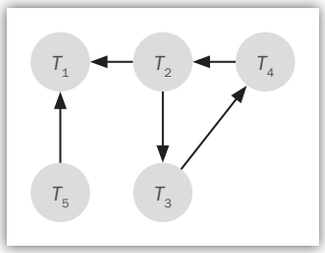


Figura 19.5. Grafo global de espera de la Figura 19.4

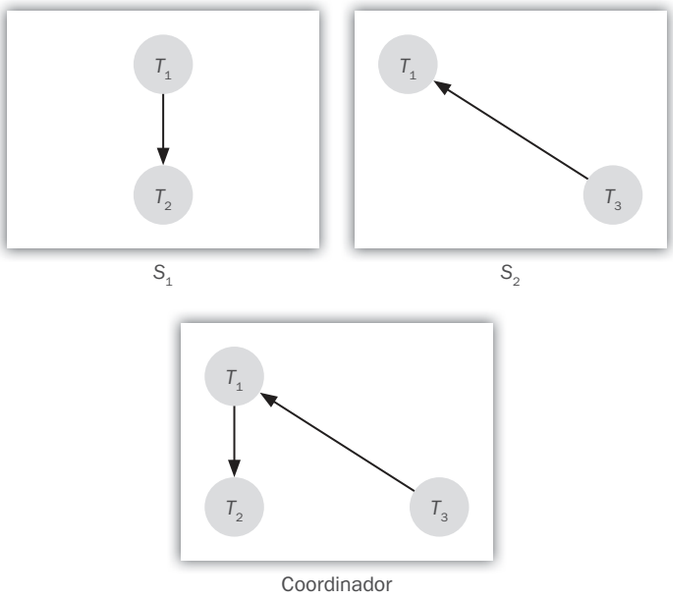


Figura 19.6. Ciclos falsos en el grafo global de espera.

Supóngase que el sitio S_1 descubre que se ha producido un fallo. Debe iniciar un procedimiento que permita que el sistema se reconfigure y continúe con el modo normal de operación:

- Si en el momento del fallo había transacciones activas en un sitio que haya fallado o quedado inaccesible, hay que abortar esas transacciones. Resulta conveniente abortarlas cuanto antes, ya que puede que tengan bloqueos sobre datos de sitios que sigan activos; puede que esperar a que el sitio que ha fallado o que ha quedado inaccesible vuelva a estar accesible impida otras transacciones en sitios que están operativos. No obstante, en algunos casos, cuando los objetos de datos están replicados, puede que sea posible seguir adelante con las operaciones de lectura y de actualización aunque algunas réplicas estén inaccesibles. En ese caso, cuando se recupera el sitio que ha fallado, si tenía réplicas de algún objeto de datos, debe obtener los valores actualizados de esos objetos de datos y asegurarse de que recibe todas las actualizaciones posteriores. Este problema se aborda en la Sección 19.6.1.
- Si los datos replicados se guardan en un sitio que ha fallado o que está inaccesible, hay que actualizar el catálogo para que las consultas no hagan referencia a la copia ubicada en ese sitio. Cuando el sitio vuelva a estar activo, hay que asegurarse de que los datos que alberga sean consistentes, como se verá en la Sección 19.6.3.
- Si el sitio que ha fallado es un servidor central de algún subsistema, hay que celebrar una *elección* para determinar el nuevo servidor (consulte la Sección 19.6.5). Entre los servidores centrales están los servidores de nombres, los coordinadores de concurrencia y los detectores globales de interbloqueos.

Dado que, en general, no es posible distinguir entre los fallos de los enlaces de red y los de los sitios, hay que diseñar todos los esquemas de reconfiguración para que funcionen de manera correcta en caso de división de la red. En concreto, deben evitarse las situaciones siguientes, para asegurar la consistencia:

- Que se elijan dos o más servidores centrales en particiones distintas.
- Que más de una partición actualice un mismo elemento de datos replicado.

Aunque las bases de datos tradicionales se centran más en la consistencia, en la actualidad existen muchas aplicaciones que valoran más la disponibilidad que la consistencia. El diseño de los protocolos de replicación es diferente en estos sistemas y se verán en la Sección 19.6.6.

19.6.1. Enfoque basado en la mayoría

Se puede modificar el enfoque basado en la mayoría del control distribuido de la concurrencia de la Sección 19.5.1.4 para que funcione a pesar de los fallos. En este enfoque, cada objeto de datos guarda con él un número de versión para detectar el momento en que se escribió en él por última vez. Siempre que una transacción escribe un objeto, actualiza también su número de versión de la siguiente forma:

- Si el objeto de datos a se replica en n sitios diferentes, se debe enviar un mensaje de solicitud de bloqueo a más de la mitad de los n sitios en los que se guarda a. La transacción no opera sobre a hasta que ha conseguido obtener un bloqueo en la mayoría de las réplicas de a.
- Las operaciones de lectura examinan todas las réplicas sobre las que se ha obtenido el bloqueo y leen el valor de la réplica que tenga el mayor número de versión (de manera opcional, también pueden escribir ese valor en las réplicas con números de versión más bajos). Las operaciones de escritura leen todas las réplicas, igual que hacen las operaciones de lectura, para hallar el número de versión más elevado (normalmente, una operación de lectura habrá llevado a cabo antes este paso de la transacción, y el resultado obtenido se puede volver a utilizar). El nuevo número de versión es una unidad mayor que el mayor número de versión encontrado. La operación de escritura escribe en todas las réplicas sobre las que ha obtenido bloqueos y da al número de versión de todas las réplicas el valor del nuevo número de versión.

Los fallos durante una transacción (tanto las divisiones de la red como los fallos de los sitios) se pueden tolerar siempre que (1) los sitios disponibles en el momento del compromiso contengan la mayoría de las réplicas de todos los objetos en los que se ha escrito y (2) durante las operaciones de lectura se lean la mayoría de las réplicas para averiguar los números de versión. Si no se cumplen estos requisitos, hay que abortar la transacción. Siempre que se satisfagan estos requisitos, se puede utilizar el protocolo de compromiso de dos fases, como siempre, en los sitios que estén disponibles.

En este esquema, la reintegración resulta trivial; no hay que hacer nada. Esto se debe a que las operaciones de escritura habrían actualizado la mayoría de las réplicas, mientras las operaciones de lectura leerán la mayoría de las réplicas y hallarán, como mínimo, una que tenga la última versión.

La técnica de numeración de las versiones utilizada con el protocolo de mayoría se puede utilizar con el protocolo de consenso de quórum en presencia de fallos. Los detalles (evidentes) se dejan al lector. No obstante, el riesgo de que los fallos eviten que el sistema procese las transacciones aumenta si se asignan pesos superiores a algunos sitios.

19.6.2. Enfoque leer uno, escribir todos los disponibles

Como caso especial de consenso de quórum se puede emplear el protocolo sesgado, si se asignan pesos unitarios a todos los sitios, se define el quórum de lectura como 1 y se define el quórum de escritura como n (todos los sitios). En este caso especial no hace falta utilizar números de versión; sin embargo, con que falle un solo sitio que contenga un elemento de datos no podrá llevarse a cabo ninguna operación de escritura en ese elemento, ya que no se dispondrá del quórum de escritura. Este protocolo se denomina protocolo **leer uno, escribir todos,** ya que hay que escribir todas las réplicas.

Para permitir que el trabajo continúe en caso de fallos, sería deseable poder utilizar el protocolo **leer uno, escribir todos los disponibles.** En este enfoque las operaciones de lectura se llevan a cabo como en el esquema **leer uno, escribir todos;** se puede leer cualquier réplica disponible, y sobre ella se obtiene un bloqueo de lectura. Se envía una operación de escritura a todas las réplicas, y se adquieren bloqueos de escritura sobre todas ellas. Si algún sitio no está disponible, el gestor de transacciones continúa su labor sin esperar a que se recupere.

Aunque este enfoque parezca muy atractivo, presenta varias complicaciones. En concreto, los fallos de comunicación temporales pueden hacer que un sitio parezca no disponible, lo que hace que la operación de escritura no se lleve a cabo pero, cuando el enlace se restaura, el sitio no sabe que tiene que llevar a cabo acciones de reintegración para ponerse al día con las operaciones de escritura que se han perdido. Además, si la red se divide, puede que cada partición actualice el mismo elemento de datos, creyendo que los sitios de las demás particiones no funcionan.

El esquema leer uno, escribir todos los disponibles puede utilizarse si nunca se producen divisiones de la red, pero puede dar lugar a inconsistencias en caso de que se produzcan.

19.6.3. Reintegración de los sitios

La reintegración al sistema de los sitios o enlaces reparados exige la adopción de precauciones. Cuando un sitio que ha fallado se recupera, debe iniciar un procedimiento para actualizar sus tablas del sistema que reflejen las modificaciones realizadas mientras estaba fuera de servicio. Si el sitio tiene réplicas de elementos de datos, debe obtener sus valores actualizados y asegurarse de que recibe todas las actualizaciones que se produzcan a partir de ese momento. La reintegración de los sitios es más complicada de lo que parece a primera vista, ya que puede que haya actualizaciones de los elementos de datos que se procesen durante el tiempo en el que el sitio se está recuperando.

Una solución sencilla es detener temporalmente todo el sistema hasta que el sitio que ha fallado vuelva a estar activo. En la mayor parte de las aplicaciones, sin embargo, esa detención temporal plantea problemas inaceptables. Se han desarrollado técnicas para permitir que los sitios que han fallado se reintegren mientras se ejecutan de manera concurrente las actualizaciones de los elementos de datos. Antes de que se conceda ningún bloqueo de lectura o escritura sobre algún elemento de datos, el sitio debe asegurarse de que se ha puesto al día con todas las actualizaciones de ese elemento de datos. Si se recupera un enlace que había fallado, se pueden volver a unir dos o más divisiones de la red. Dado que la división de la red limita las operaciones admisibles para algunos de los sitios, o para todos ellos, hay que informar con prontitud a todos los sitios de la recuperación de ese enlace. Consulte las notas bibliográficas para obtener más información sobre la recuperación en los sistemas distribuidos.

19.6.4. Comparación con la copia de seguridad remota

Los sistemas de copia de seguridad remota, que se estudiaron en la Sección 16.9, y la réplica en las bases de datos distribuidas son dos enfoques alternativos para la provisión de una disponibilidad elevada. La diferencia principal entre los dos esquemas es que, con los sistemas de copia de seguridad remota, las acciones como el control de concurrencia y la recuperación se llevan a cabo en un único sitio, y solo se replican en el otro sitio los datos y los registros del registro histórico. En concreto, los sistemas de copia de seguridad remota ayudan a evitar el compromiso de dos fases, y las sobrecargas resultantes. Además, las transacciones solo tienen que comunicarse a un solo sitio (el sitio principal), y así se evita la sobrecarga de la ejecución del código de las transacciones entre varios sitios. Por tanto, los sistemas de copia de seguridad remota ofrecen un enfoque de la elevada disponibilidad de menor coste que las réplicas.

Por otra parte, la réplica puede ofrecer mayor disponibilidad al tener disponibles varias réplicas y utilizar el protocolo de mayoría.

19.6.5. Selección del coordinador

Varios de los algoritmos que se han presentado exigen el empleo de un coordinador. Si el coordinador falla por problemas en el sitio en el que reside el sistema, el sistema solo puede continuar la ejecución reiniciando un nuevo coordinador en otro sitio. Un modo de continuar la ejecución es mantener una copia de seguridad en un coordinador suplente, que esté preparado para asumir la responsabilidad si el coordinador falla.

El **coordinador suplente** es un sitio que, además de otras tareas, mantiene de manera local suficiente información como para poder asumir el papel de coordinador con una interrupción mínima del sistema distribuido. Tanto el coordinador como su suplente reciben todos los mensajes dirigidos al coordinador. El coordinador suplente ejecuta los mismos algoritmos y mantiene la misma información interna de estado (como, por ejemplo, el coordinador de concurrencia o la tabla de bloqueos) que el coordinador real. La única diferencia de funcionamiento entre el coordinador y su suplente es que el suplente no emprende ninguna acción que afecte a otros sitios. Esas acciones se dejan al coordinador real.

En caso de que el coordinador suplente detecte el fallo del coordinador real, asume el papel de coordinador. Dado que el suplente dispone de toda la información que tenía el coordinador que ha fallado, el procesamiento puede continuar sin interrupción.

La ventaja principal del enfoque del suplente es la posibilidad de continuar el procesamiento de manera inmediata. Si no hubiera un suplente dispuesto a asumir la responsabilidad del coordinador, el coordinador que se designara *ex novo* tendría que buscar la información en todos los sitios del sistema para poder ejecutar las tareas de coordinación. Con frecuencia, la única fuente para obtener parte de la información necesaria es el coordinador que ha fallado. En ese caso, puede que sea necesario abortar parte de las transacciones activas (o todas ellas) y reiniciarlas bajo el control del nuevo coordinador.

Por tanto, el enfoque del coordinador suplente evita retrasos sustanciales mientras el sistema distribuido se recupera del fallo del coordinador. El inconveniente es la sobrecarga de la ejecución duplicada de las tareas del coordinador. Además, el coordinador y su suplente necesitan comunicarse de manera regular para asegurarse de que sus actividades están sincronizadas.

En resumen, el enfoque del coordinador suplente supone una sobrecarga durante el procesamiento normal para permitir una recuperación rápida de los fallos del coordinador.

A falta de un coordinador suplente designado, o con objeto de tratar varios fallos, los sitios que siguen funcionando pueden escoger de manera dinámica un nuevo coordinador. Los **algoritmos de elección** permiten que los sitios escojan el sitio del nuevo coordinador de manera descentralizada. Los algoritmos de selección necesitan que se asocie un número de identificación único con cada sitio activo del sistema.

El **algoritmo de acoso** para la elección funciona de la siguiente forma: para no complicar la notación ni la descripción, suponga que el número de identificación del sitio S_i es i y que el coordinador elegido siempre será el sitio activo con el número de identificación más elevado. Por tanto, cuando un coordinador falla, el algoritmo debe elegir el sitio activo que tenga el número de identificación más elevado. El algoritmo debe enviar ese número a cada sitio activo del sistema. Además, el algoritmo debe proporcionar un mecanismo por el que los sitios que se recuperen de un fallo puedan identificar al coordinador activo. Suponga que el sitio S_i envía una solicitud que el coordinador no responde dentro del intervalo de tiempo predeterminado T. En esa situación se supone que el coordinador ha fallado y S_i intenta elegirse a sí mismo como sitio del nuevo coordinador.

El sitio S_i envía un mensaje de elección a cada sitio que tenga un número de identificación más elevado. Luego espera, un intervalo de tiempo T, la respuesta de cualquiera de esos sitios. Si no recibe respuesta dentro del tiempo T, da por supuesto que todos los sitios con números mayores que i han fallado, se elige a sí mismo sitio del nuevo coordinador y envía un mensaje para informar a todos los sitios activos con números de identificación menores de que i es el sitio en el que reside el nuevo coordinador.

Si S_i recibe respuesta, comienza un intervalo de tiempo T' para recibir un mensaje que lo informe de que se ha elegido un sitio con un número de identificación más elevado (algún otro sitio se está eligiendo coordinador, y debe comunicar los resultados dentro del tiempo T'). Si S_i no recibe ningún mensaje antes de T', da por supuesto que el sitio con el número más elevado ha fallado y vuelve a iniciar el algoritmo.

Después de que un sitio que ha fallado se haya recuperado, comienza de inmediato la ejecución de ese mismo algoritmo. Si no hay ningún sitio activo con un número más elevado, el sitio que se ha recuperado obliga a todos los sitios con números más bajos a permitirle transformarse en el sitio coordinador, aunque ya haya un coordinador activo con un número más bajo. Por este motivo, al algoritmo se le denomina algoritmo de *acoso*. Si la red se divide, el algoritmo de acoso elige un coordinador diferente en cada partición; para garantizar que, a lo sumo, se elige un coordinador, los sitios que ganan deben comprobar también que la mayoría de los sitios se hallan en su partición.

19.6.6. Compromiso entre consistencia y disponibilidad

Los protocolos que se han visto hasta el momento requieren que haya una mayoría de sitios (con pesos) en la partición para que se realicen las actualizaciones. Los sitios que se encuentran en una división minoritaria no procesan las actualizaciones; si un fallo de red genera más de dos divisiones de red, puede que ninguna tenga la mayoría de los sitios. En esta situación, el sistema quedaría sin poder realizar actualizaciones y, dependiendo del quórum de lectura, puede que incluso no puedan realizarse lecturas. El protocolo escribir todos los disponibles que se describió anteriormente proporciona esa disponibilidad pero no la consistencia.

Idealmente, se desearía disponer tanto de consistencia como de disponibilidad, incluso si se producen divisiones de red. Desafortunadamente, no es posible, un hecho que se concreta en el llamado **teorema CAP,** que indica que dado un sistema distribuido puede conseguir como mucho dos de las siguientes tres propiedades:

- Consistencia (*Consistency*).
- Disponibilidad (*Availability*).
- Tolerancia a divisiones (*Partition tolerance*).

La demostración del teorema CAP usa la siguiente definición de consistencia, con datos replicados: se dice que una ejecución de un conjunto de operaciones (lectura y escritura) es **consistente** sobre datos replicados si su resultado es el mismo que si las operaciones se realizasen en un único sitio, en cierto orden secuencial, y el orden secuencial es consistente con la ordenación de las operaciones que se realizan en cada uno de los procesos (transacciones).

La noción de consistencia es similar a la de atomicidad de las transacciones, pero en ella cada operación se trata como una transacción y es más débil que la propiedad de atomicidad.

En un sistema distribuido de gran tamaño, no se pueden evitar divisiones y, por tanto, hay que sacrificar o la consistencia o la disponibilidad. Los esquemas que se han descrito anteriormente sacrifican la disponibilidad por la consistencia cuando se producen divisiones.

Suponga un sistema de red social en la web que replica sus datos en tres servidores y se produce una división de red que evita que los servidores se comuniquen entre sí. Como ninguna de las particiones tiene la mayoría, no sería posible ejecutar actualizaciones en ninguna de las divisiones. Si uno de los servidores se encuentra en la misma división que un usuario, el usuario tiene acceso a los datos, pero no podría actualizarlos, ya que otro usuario podría estar actualizando concurrentemente el mismo objeto desde otra división, lo que podría generar inconsistencias. Las inconsistencias no son tan graves en un sistema de una red social como en una base de datos bancaria. El diseñador del sistema puede decidir que un usuario que tenga acceso al sistema debería poder realizar actualizaciones en cualquiera de las réplicas que estén accesibles, incluso con el riesgo de generar inconsistencias.

En contraste con los sistemas de bases de datos bancarias que requieren cumplir las propiedades ACID, los sistemas como los de redes sociales indicadas requieren que se cumplan las propiedades **BASE:**

- Básicamente disponibles (*Basically Available*).
- Estado difuso (*Soft state*).
- Eventualmente consistente (*Eventually consistent*).

El principal requisito es la disponibilidad, incluso a costa de la consistencia. Se deberían permitir las actualizaciones, incluso aunque se produzcan divisiones, siguiendo el ejemplo del protocolo escribir todos los disponibles (que es similar a la replicación multimaestro descrita en la Sección 19.5.3). El estado difuso se refiere a la propiedad de que el estado de la base de datos puede que no esté definido con precisión y cada réplica pueda tener un estado algo diferente debido a la división de la red. Eventualmente consistente hace referencia al requisito de que si la división se resuelve, eventualmente todas las réplicas conseguirán ser consistentes entre sí.

El último paso requiere identificar las copias inconsistentes de los elementos de datos; si uno de ellos es de una versión anterior, la versión más antigua se sustituye por la nueva. Sin embargo, puede ocurrir que las dos copias se generen en actualizaciones independientes sobre una copia base común. El esquema para detectar estas actualizaciones inconsistentes se denomina esquema de vector de versión y se describe en la Sección 25.5.4.

La recuperación de la consistencia si se producen actualizaciones inconsistentes requiere que las actualizaciones se mezclen de alguna forma que tenga sentido para la aplicación. Este paso no lo pueden manejar las bases de datos; en vez de ello la base de datos detecta el problema e informa de la inconsistencia a la aplicación y esta decide cómo resolverla.

19.7. Procesamiento distribuido de consultas

En el Capítulo 13 se estudió que existen una gran variedad de métodos para el cálculo de la respuesta a una consulta. Se examinaron varias técnicas para escoger una estrategia de procesamiento de consultas que minimice el tiempo que se tarda en calcular la respuesta. Para los sistemas centralizados, el criterio principal para medir el coste de una estrategia dada es el número de accesos a disco. En los sistemas distribuidos hay que tener en cuenta otros aspectos, entre los que se incluyen:

- El coste de la transmisión de los datos por la red.
- La posible ganancia de rendimiento al hacer que varios sitios procesen en paralelo diferentes partes de la consulta.

El coste relativo de la transferencia de datos por la red y del intercambio de datos con el disco varía ampliamente en función del tipo de red y de la velocidad de los discos. Por tanto, en general, uno no se puede centrar exclusivamente en los costes de disco ni en los de la red. Más bien hay que hallar un buen equilibrio entre los dos.

19.7.1. Transformación de consultas

Considere una consulta extremadamente sencilla: «Hallar todas las tuplas de la relación *cuenta*». Aunque la consulta es sencilla (en realidad, trivial), su procesamiento no lo es, ya que puede que la relación *cuenta* esté fragmentada, replicada, o ambas cosas, como se vio en la Sección 19.2. Si la relación *cuenta* está replicada, hay que elegir la réplica. Si las réplicas no se han dividido, se escoge aquella para la que el coste de transmisión sea mínimo. Sin embargo, si alguna réplica se ha dividido, la elección no resulta tan sencilla, ya que hay que calcular varias reuniones o uniones para reconstruir la relación *cuenta*. En ese caso, el número de estrategias para el sencillo ejemplo escogido puede ser elevado. Puede que la optimización de las consultas mediante la enumeración exhaustiva de todas las estrategias alternativas no resulte práctica en esas situaciones.

La transparencia de la fragmentación implica que los usuarios pueden escribir una consulta como:

$$\sigma_{nombre_sucursal\ =\ \text{«Hillside»}}\ (cuenta)$$

Ya que *cuenta* está definida como:

$$cuenta_1 \cup cuenta_2$$

la expresión que resulta del esquema de traducción de nombres es:

$$\sigma_{nombre_sucursal\ =\ \text{«Hillside»}}\ (cuenta_1 \cup cuenta_2)$$

Esta expresión se puede simplificar de manera automática mediante las técnicas de optimización de consultas del Capítulo 13. El resultado es la expresión:

$$\sigma_{nombre_sucursal\ =\ \text{«Hillside»}}\ (cuenta_1) \cup \sigma_{nombre_sucursal\ =\ \text{«Hillside»}}\ (cuenta_2)$$

que incluye dos subexpresiones. La primera solo implica a $cuenta_1$ y, por tanto, puede evaluarse en el sitio Hillside. La segunda solo implica a $cuenta_2$ y, por tanto, puede evaluarse en el sitio Valleyview.

Hay otra optimización más que puede hacerse al evaluar:

$$\sigma_{nombre_sucursal\ =\ \text{«Hillside»}}\ (cuenta_1)$$

Dado que $cuenta_1$ solo contiene tuplas correspondientes a la sucursal de Hillside, se puede eliminar la operación de selección. Al evaluar:

$$\sigma_{nombre_sucursal\ =\ \text{«Hillside»}}\ (cuenta_2)$$

se puede aplicar la definición del fragmento de $cuenta_2$ para obtener:

$$\sigma_{nombre_sucursal\ =\ \text{«Hillside»}}\ (\sigma_{nombre_sucursal\ =\ \text{«Valleyview»}}\ (cuenta))$$

Esta expresión es el conjunto vacío, independientemente del contenido de la relación *cuenta*.

Por tanto, la estrategia final es que el sitio Hillside devuelva $cuenta_1$ como resultado de la consulta.

19.7.2. Procesamiento de reuniones sencillas

Como se vio en el Capítulo 13, una decisión importante en la selección de una estrategia de procesamiento de consultas es la elección de la estrategia de reunión. Considere la siguiente expresión del álgebra relacional:

$$cuenta \bowtie depositante \bowtie sucursal$$

Suponga que ninguna de las tres relaciones está replicada ni fragmentada, y que *cuenta* está almacenada en el sitio S_1, *depositante* en S_2 y *sucursal* en S_3. Suponga que S_F indica el sitio en el que se ha formulado la consulta. El sistema necesita obtener el resultado en el sitio S_F. Entre las posibles estrategias para el procesamiento de esta consulta figuran las siguientes:

- Enviar copias de las tres relaciones al sitio S_F. Empleando las técnicas del Capítulo 13, hay que escoger una estrategia para el procesamiento local de toda la consulta en el sitio S_F.
- Enviar una copia de la relación *cuenta* al sitio S_2 y calcular $temp_1 = cuenta \bowtie impositor$ en S_2. Enviar $temp_1$ de S_2 a S_3 y calcular $temp_2 = temp_1 \bowtie sucursal$ en S_3. Enviar el resultado $temp_2$ a S_F.
- Diseñar estrategias parecidas a la anterior con los roles de S_1, S_2 y S_3 intercambiados.

Ninguna de las estrategias es la mejor en todos los casos. Entre los factores que deben tenerse en cuenta están el volumen de los datos que se envían, el coste de la transmisión de los bloques de datos entre cada par de sitios y la velocidad relativa de procesamiento en cada sitio. Considere las dos primeras estrategias mencionadas. Suponga que los índices existentes en S_2 y en S_3 resultan útiles para calcular la reunión. Si se envían las tres relaciones a S_F, habrá que volver a crear esos índices en S_F o emplear una estrategia de reunión diferente, posiblemente más costosa. Esta recreación de los índices supone una sobrecarga adicional de procesamiento y más accesos a disco. Con la segunda estrategia hay que enviar una relación potencialmente grande ($cuenta \bowtie depositante$) de S_2 a S_3. Esta relación repite el nombre de cada cliente una vez por cada cuenta que tenga abierta. Por tanto, puede que la segunda estrategia, en comparación con la primera, dé lugar a un mayor volumen de transmisión de datos por la red.

19.7.3. Estrategia de semirreunión

Suponga que se desea evaluar la expresión $r_1 \bowtie r_2$, donde r_1 y r_2 se almacenan en los sitios S_1 y S_2, respectivamente. Sean R_1 y R_2 los esquemas de r_1 y de r_2. Supóngase que se desea obtener el resultado en S_1. Si hay muchas tuplas de r_2 que no se reúnen con ninguna tupla de r_1, el envío de r_2 a S_1 supone el envío de tuplas que no contribuyen al resultado. Se desea eliminar esas tuplas antes de enviar los datos a S_1, especialmente si los costes de la red son elevados.

Una posible estrategia para lograr todo esto es la siguiente:

1. Calcular $temp_1 \leftarrow \Pi_{R_1 \cap R_2}(r_1)$ en S_1.
2. Enviar $temp_1$ de S_1 a S_2.
3. Calcular $temp_2 \leftarrow r_2 \bowtie temp_1$ en S_2.
4. Enviar $temp_2$ de S_2 a S_1.
5. Calcular $r_1 \bowtie temp_2$ en S_1. La relación resultante es la misma que $r_1 \bowtie r_2$.

Antes de considerar la eficiencia de esta estrategia hay que comprobar que la estrategia calcula la respuesta correcta. En el paso 3 $temp_2$ tiene el resultado de $r_2 \bowtie \Pi_{R_1 \cap R_2}(r_1)$. En el paso 5 se calcula:

$$r_1 \bowtie r_2 \bowtie \Pi_{R_1 \cap R_2}(r_1)$$

Dado que la reunión es asociativa y conmutativa, se puede volver a escribir esta expresión como:

$$(r_1 \bowtie \Pi_{R_1 \cap R_2}(r_1)) \bowtie r_2$$

Dado que $r_1 \bowtie \Pi(_{R_1 \cap R_2})(r_1) = r_1$, la expresión es, realmente, igual a $r_1 \bowtie r_2$, la expresión que se pretendía evaluar.

Esta estrategia resulta especialmente ventajosa cuando contribuyen a la reunión relativamente pocas tuplas de r_2. Es probable que se produzca esta situación si r_1 es resultado de una expresión de álgebra relacional que implica una selección. En esos casos, puede que $temp_2$ tenga significativamente menos tuplas que r_2. El ahorro de costes de la estrategia procede de no tener que enviar a S_1 más que $temp_2$, en vez de toda r_2. El envío de $temp_1$ a S_2 supone un coste adicional. Si solo contribuye a la reunión una fracción de tuplas de r_2 lo bastante pequeña, la sobrecarga del envío de $temp_1$ queda dominada por el ahorro de no tener que enviar más que una parte de las tuplas de r_2.

Esta estrategia se denomina **estrategia de semirreunión** (debido al operador de semirreunión del álgebra relacional, denotado por $\ltimes$). La semirreunión de r_1 con r_2, denotada por $r_1 \ltimes r_2$, es:

$$\Pi_{R_1}(r_1 \bowtie r_2)$$

Por tanto, $r_1 \ltimes r_2$ selecciona las tuplas de r_1 que han contribuido a $r_1 \bowtie r_2$. En el paso 3 $temp_2 = r_2 \ltimes r_1$.

Para las reuniones de varias relaciones, esta estrategia puede ampliarse a una serie de pasos de semirreunión. Se ha desarrollado un importante corpus teórico en relación con el empleo de la semirreunión para la optimización de consultas. Parte de esta teoría se menciona en las notas bibliográficas.

19.7.4. Estrategias de reunión que aprovechan el paralelismo

Considere una reunión de cuatro relaciones:

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$$

donde la relación r_i se guarda en el sitio S_i. Suponga que el resultado debe presentarse en el sitio S_1. Hay muchas estrategias posibles para la evaluación en paralelo (el problema del procesamiento de las consultas en paralelo se estudió con detalle en el Capítulo 18). En una de estas estrategias, r_1 se envía a S_2, donde se calcula $r_1 \bowtie r_2$. Al mismo tiempo, r_3 se envía a S_4, donde se calcula $r_3 \bowtie r_4$. El sitio S_2 puede enviar las tuplas de $(r_1 \bowtie r_2)$ a S_1 a medida que se generan, en lugar de esperar a que se calcule toda la reunión. De manera parecida, S_4 puede enviar las tuplas de $(r_3 \bowtie r_4)$ a S_1. Una vez que las tuplas de $(r_1 \bowtie r_2)$ y de $(r_3 \bowtie r_4)$ hayan llegado a S_1, puede comenzar el cálculo de $(r_1 \bowtie r_2) \bowtie (r_3 \bowtie r_4)$, con la técnica de reunión canalizada de la Sección 12.7.2.2. Por tanto, el cálculo del resultado de la reunión final en S_1 puede hacerse en paralelo con el cálculo de $(r_1 \bowtie r_2)$ en S_2, y con el de $(r_3 \bowtie r_4)$ en S_4.

19.8. Bases de datos distribuidas heterogéneas

Muchas de las últimas aplicaciones de bases de datos necesitan datos de gran variedad de bases de datos ya existentes y ubicadas en un conjunto heterogéneo de entornos de hardware y de software. El tratamiento de la información ubicada en bases de datos distribuidas heterogéneas exige una capa de software adicional por encima de los sistemas de bases de datos ya existentes. Esta capa de software se denomina **sistema de bases de datos múltiples.** Puede que los sistemas locales de bases de datos empleen modelos lógicos y lenguajes de definición y de tratamiento de datos diferentes, y que difieran en sus mecanismos de control de concurrencia y de administración de las transacciones. Los sistemas de bases de datos múltiples crean la ilusión de la integración lógica de las bases de datos sin necesidad de su integración física.

La integración completa de sistemas heterogéneos en una misma base de datos distribuida homogénea suele resultar difícil o imposible:

- **Dificultades técnicas.** La inversión en los programas de aplicaciones basados en los sistemas de bases de datos ya existentes puede ser enorme, y el coste de transformar esas aplicaciones puede resultar prohibitivo.
- **Dificultades organizativas.** Aunque la integración resulte *técnicamente* posible, puede que no lo sea *políticamente*, porque los sistemas de bases de datos ya existentes pertenezcan a diferentes empresas u organizaciones. En ese caso es importante que el sistema de bases de datos múltiples permita que los sistemas de bases de datos locales conserven un elevado grado de **autonomía** para la base de datos local y para las transacciones que se ejecuten con esos datos.

Por estos motivos, los sistemas de bases de datos múltiples ofrecen ventajas significativas que compensan la sobrecarga que suponen. En esta sección se proporciona una visión general de los retos que se afrontan al construir entornos con bases de datos múltiples desde el punto de vista de la definición de los datos y del procesamiento de las consultas.

19.8.1. Vista unificada de los datos

Cada sistema local de administración de bases de datos puede utilizar un modelo de datos diferente. Por ejemplo, puede que algunos empleen el modelo relacional, mientras que otros pueden emplear modelos de datos más antiguos, como el de red o el jerárquico.

Dado que se supone que los sistemas con bases de datos múltiples ofrecen la ilusión de un solo sistema de bases de datos integrado, hay que utilizar un modelo de datos común. Una opción adoptada con frecuencia es el modelo relacional, con SQL como lenguaje común de consulta. En realidad, hoy en día hay varios sistemas disponibles que permiten realizar consultas SQL en sistemas de administración de bases de datos no relacionales.

Otra dificultad es proporcionar un esquema conceptual común. Cada sistema local ofrece su propio esquema conceptual. El sistema de bases de datos múltiples debe integrar esos esquemas independientes en uno común. La integración de los esquemas es una tarea complicada, sobre todo por la heterogeneidad semántica.

La integración de los esquemas no es la mera traducción directa de unos lenguajes de definición de datos a otros. Puede que aparezcan los mismos nombres de atributos en diferentes bases de datos locales pero con significados distintos. Puede que los tipos de datos utilizados en un sistema no estén soportados por los demás y que la traducción de unos tipos a otros no resulte sencilla. Incluso en el caso de tipos de datos idénticos, pueden surgir problemas debidos a la representación física de los datos: puede que un sistema utilice ASCII de 8 bits, otro ASCII de 16 bits y otro EBCDIC; las representaciones en coma flotante pueden ser diferentes, los enteros pueden representarse como *orden más significativo (big-endian)* o como *orden menos significativo (little-endian)*. En el nivel semántico, el valor entero de una longitud puede ser pulgadas en un sistema y milímetros en otro, lo que crea una situación incómoda en la que la igualdad entre los enteros sea solo un concepto aproximado (como ocurre siempre con los números con coma flotante). Puede que el mismo nombre aparezca en idiomas distintos en diferentes sistemas. Por ejemplo, puede que un sistema basado en Estados Unidos se refiera a la ciudad de Saragossa, mientras que uno con base en España lo haga como Zaragoza.

Todas estas diferencias aparentemente menores deben registrarse de manera adecuada en el esquema conceptual global común. Hay que proporcionar funciones de traducción. Hay que anotar los índices para el comportamiento dependiente del sistema (por ejemplo, el orden de clasificación de los caracteres no alfanuméricos no es igual en ASCII que en EBCDIC). Como ya se ha comentado, puede que la alternativa de convertir cada base de datos a un formato común no resulte factible sin dejar obsoletos los programas de aplicación ya existentes.

19.8.2. Procesamiento de las consultas

El procesamiento de las consultas en las bases de datos heterogéneas puede resultar complicado. Algunos de los problemas son:

- Dada una consulta en un esquema global, puede que haya que traducir la consulta a consultas en los esquemas locales de cada uno de los sitios en que hay que ejecutar la consulta. Hay que volver a traducir los resultados de las consultas al esquema global.

  La tarea se simplifica escribiendo **envolturas** (*wrappers*) para cada origen de datos, que ofrezcan una vista de los datos locales en el esquema global. Las envolturas también traducen las consultas del esquema global a consultas del esquema local y vuelven a traducir los resultados al esquema global. Las envolturas puede ofrecerlas cada sitio o escribirse de manera independiente como parte del sistema de bases de datos múltiples.

  Las envolturas pueden, incluso, utilizarse para proporcionar una vista relacional de orígenes de datos no relacionales, como las páginas web (posiblemente con interfaces de formularios), archivos planos, bases de datos jerárquicas y de red y sistemas de directorio.
- Puede que algunos orígenes de datos solo ofrezcan capacidades de consulta limitadas; por ejemplo, puede que soporten selecciones pero no reuniones. Puede incluso que restrinjan la forma de las selecciones, permitiéndolas solo para determinados campos; los orígenes de datos web con interfaces de formulario son un ejemplo de este tipo de orígenes de datos. Por tanto, puede que haya que dividir las consultas para que se lleven a cabo en parte en el origen de datos y en parte en el sitio que formula la consulta.
- En general, puede que haya que acceder a más de un sitio para responder a una consulta dada. Es posible que haya que procesar las respuestas obtenidas de los diferentes sitios para eliminar los valores duplicados. Suponga que un sitio contiene las tuplas de *cuenta* que satisfacen la selección $saldo < 100$, mientras que otro contiene las que satisfacen $saldo > 50$. Una consulta sobre toda la relación *cuenta* exigiría acceder a los dos sitios y eliminar las respuestas duplicadas consecuencia de las tuplas con saldo entre 50 y 100, que están replicadas en los dos sitios.
- La optimización global de consultas en bases de datos heterogéneas resulta difícil, ya que puede que el sistema de ejecución de consultas no conozca los costes de los planes de consulta alternativos en los diferentes sitios. La solución habitual es confiar solo en la optimización a nivel local y utilizar únicamente la heurística a nivel global.

Los sistemas **mediadores** son sistemas que integran varios orígenes de datos heterogéneos, lo que proporciona una vista global integrada de los datos y facilidades de consulta sobre la misma. A diferencia de los sistemas de bases de datos múltiples completos, los sistemas mediadores no se ocupan del procesamiento de las transacciones (los términos mediador y bases de datos múltiples suelen utilizarse de manera indistinta, y puede que los sistemas denominados mediadores soporten formas limitadas de transacción). El término **base de datos virtual** se utiliza para hacer referencia a los sistemas de bases de datos múltiples o a los sistemas mediadores, ya que ofrecen la apariencia de una sola base de datos con un esquema global, aunque los datos estén en varios sitios en esquemas locales.

19.8.3. Gestión de transacciones en bases de datos múltiples

Un sistema de bases de datos múltiples admite dos tipos de transacciones:

1. **Transacciones locales.** Estas transacciones las ejecuta cada sistema de bases de datos local fuera del control del sistema de bases de datos múltiples.
2. **Transacciones globales.** Estas transacciones se ejecutan bajo el control del sistema de bases de datos múltiples.

El sistema de bases de datos múltiples conoce que las transacciones locales se ejecutan en los sitios locales, pero no conoce qué transacciones en concreto se están ejecutando, o a qué datos pueden acceder.

Para asegurar la autonomía de cada uno de los sistemas de bases de datos local estos requieren que no se realicen cambios en su software. Un sistema de bases de datos de un sitio, por tanto, no se puede comunicar directamente con otro de otro sitio para sincronizar la ejecución de una transacción global activa en varios sitios.

Como el sistema de bases de datos múltiple no tiene control sobre la ejecución de las transacciones locales, cada sistema local debe usar un esquema de control de concurrencia (por ejemplo, el bloqueo de dos fases o las marcas de tiempo), para asegurar que su planificación es secuenciable. Además, en caso de bloqueo, el sistema local debe prevenir contra posibles interbloqueos locales.

La garantía de secuencialidad local no es suficiente para asegurar la secuencialidad global. Como ejemplo, considere dos transacciones globales T_1 y T_2, cada una de ellas accede y actualiza dos elementos de datos A y B, ubicados en los sitios S_1 y S_2, respectivamente. Suponga que las planificaciones locales son secuenciables. Sigue existiendo una situación en la que en el sitio S_1, T_2 sigue a T_1, mientras que en el sitio S_2, T_1 sigue a T_2, produciendo una planificación global no secuenciable. De hecho, incluso sin que se produzca concurrencia entre las transacciones globales (es decir, una transacción global solo se envía después de que la anterior haya pasado a comprometida o abortada), la secuencialidad local no es suficiente para asegurar la secuencialidad global (consulte el Ejercicio práctico 19.14).

Dependiendo de la implementación de los sistemas de bases de datos locales, una transacción global puede que no pueda controlar el comportamiento de los bloqueos concretos de las subtransacciones locales. Por tanto, aunque todos los sistemas locales sigan el bloqueo de dos fases, puede que solo se pueda asegurar que las transacciones locales cumplen con las reglas de protocolo. Por ejemplo, un sistema de bases de datos local puede comprometer su subtransacción y liberar los bloqueos, mientras que la subtransacción en otro sistema local todavía se esté ejecutando. Si los sistemas locales permiten controlar el comportamiento de los bloqueos y todos los sistemas siguen el bloqueo de dos fases, entonces el sistema de bases de datos múltiples puede asegurar que las transacciones globales usan bloqueos en la forma de dos fases y los puntos de bloqueo de las transacciones en conflicto definirían su orden de secuencialidad global. Si distintos sistemas locales utilizan sistemas de control de concurrencia diferentes, esta ordenación inmediata del control global no funciona.

Existen muchos protocolos para asegurar la consistencia en los sistemas de bases de datos múltiples independientemente de la ejecución concurrente de transacciones globales y locales. Algunos se basan en imponer las suficientes condiciones para asegurar la secuencialidad global. Otros solo aseguran una forma de consistencia más débil que la secuencialidad, pero consiguen la consistencia con medios menos restrictivos. En la Sección 26.6 se describen enfoques para conseguir consistencia sin secuencialidad; se pueden encontrar otros enfoques en las notas bibliográficas.

Los primeros sistemas de bases de datos múltiples restringían las transacciones globales a que fuesen de solo lectura. Por tanto, evitaban la posibilidad de transacciones globales que introdujesen inconsistencias en los datos, pero no eran suficientemente restrictivas para asegurar la secuencialidad. De hecho, se pueden conseguir planificaciones globales y desarrollar un esquema que asegure la secuencialidad global, y le pedimos que consiga ambas cosas en el Ejercicio práctico 19.15.

Existe un cierto número de esquemas generales que aseguran la secuencialidad global en un entorno en el que se pueden ejecutar transacciones de solo lectura y de actualización. Varios de estos esquemas se basan en la idea de un billete. Se crea un elemento de datos especial denominado **billete** en cada uno de los sistemas de bases de datos locales. Las transacciones globales que acceden a datos de un sitio deben escribir en el billete de ese sitio. Este requisito asegura que las transacciones globales entran en conflicto directo en los sitios que visitan. Más aún, el gestor de transacciones globales puede controlar el orden en que se secuencian las transacciones globales, controlando el orden en que se accede a los billetes. En las notas bibliográficas puede encontrar referencias a estos esquemas.

Si se desea asegurar la secuencialidad global en un entorno en el que no se generan conflictos locales en cada uno de los sitios, se deben realizar algunas suposiciones sobre las planificaciones permitidas por los sistemas de bases de datos locales. Por ejemplo, si las planificaciones locales son de tal forma que el orden de compromiso y el orden de secuenciación son siempre idénticos, se puede asegurar la secuencialidad controlando el orden en que las transacciones pasan al estado de comprometida.

Un problema relacionado en los sistemas de bases de datos múltiple es el de los compromisos atómicos globales. Si todos los sistemas locales siguen el protocolo de compromiso de dos fases, se puede usar este protocolo para conseguir atomicidad global. Sin embargo, los sistemas locales que no fueron diseñados para formar parte de un sistema distribuido puede que no sean capaces de participar en este protocolo. Aunque un sistema local sea capaz de utilizar el compromiso de dos fases, la organización propietaria del sistema puede que no esté dispuesta a permitir la espera en el caso de que se produzcan bloqueos. En estos casos, hay que llegar a un compromiso entre permitir que no haya atomicidad en algunos modos de fallo. En la literatura puede encontrar más descripciones sobre estos temas (consulte las notas bibliográficas).

19.9. Bases de datos en la nube

El concepto de **computación en la nube** es relativamente nuevo en computación, surgiendo a finales de los años noventa y principios de la década siguiente, primero bajo el nombre de *software como servicio*. Los vendedores iniciales de servicios software proporcionaban aplicaciones personalizables que estaban alojadas en sus propias máquinas. El concepto de computación en la nube se desarrolló cuando los vendedores empezaron a ofrecer computadoras genéricas como un servicio en el que los clientes podían ejecutar aplicaciones software de su elección. Un cliente puede llegar a un acuerdo con un vendedor de computación en la nube para conseguir un cierto número de máquinas de una cierta capacidad, así como con cierta capacidad de almacenamiento. Tanto el número de máquinas como la capacidad de almacenamiento pueden crecer o reducirse según se necesite. Además de proporcionar servicios de computación, muchos vendedores también proporcionan otros servicios como almacenamiento de datos, servicios de mapas y otros a los que se pueda acceder usando interfaces de programación de aplicaciones de servicios web.

Para muchas empresas este modelo de computación y servicios en la nube tiene muchas ventajas. Les permite ahorrarse el personal necesario para mantener un gran sistema de soporte y acometer nuevas iniciativas empresariales sin tener que realizar con antelación un gran desembolso en sistemas de computación. Más aún, si las necesidades de la empresa crecen, se pueden añadir (computación y almacenamiento) más recursos cuando se necesiten; el vendedor de la computación en la nube suele disponer de grandes bancos de computadoras, lo que le permite responder a la demanda de asignación de recursos.

Existen distintos vendedores de servicios en la nube. Entre ellos se encuentran vendedores tradicionales de computación, así como compañías como Amazon y Google, que están buscando cómo aprovechar las grandes infraestructuras que tienen para su negocio principal.

Las aplicaciones web que necesitan guardar y obtener datos de un gran número de usuarios (que van desde millones a cientos de millones) se han convertido en un catalizador de las bases de datos en la nube. Las necesidades de estas aplicaciones son diferentes de las que tienen las aplicaciones de bases de datos tradicionales, ya que priman la disponibilidad y ampliabilidad sobre la consistencia. En los últimos años se han desarrollado un buen número de sistemas de almacenamiento de datos en la nube para dar servicio a estas aplicaciones. Se tratarán los temas de construcción de estos sistemas de almacenamiento de datos en la nube en la Sección 19.9.1.

En la Sección 19.9.2 se consideran los elementos de ejecución de los sistemas tradicionales de bases de datos en la nube. Estos sistemas tienen características tanto de los sistemas homogéneos como heterogéneos. Aunque los datos pertenecen a una organización (el cliente) y forman parte de una base de datos distribuida unificada, las computadoras subyacentes pertenecen y son gestionadas por otra organización (el vendedor del servicio). Las computadoras se encuentran remotas al cliente y se accede a ellas por Internet. En este sentido, se mantienen algunos de los desafíos de los sistemas distribuidos heterogéneos, en particular los concernientes al procesamiento de transacciones. Sin embargo, se solventan muchos de los desafíos políticos y organizativos de los sistemas heterogéneos.

Finalmente, en la Sección 19.9.3, se tratan distintos aspectos técnicos y no técnicos que afrontan en la actualidad los sistemas de bases de datos en la nube.

19.9.1. Sistemas de almacenamiento de datos en la nube

Las aplicaciones en la web tienen unos requisitos extremadamente altos de ampliabilidad. Las aplicaciones más populares tienen cientos de millones de usuarios, y muchas aplicaciones han visto cómo se incrementaba su carga varias veces en un solo año, o incluso en unos meses. Para manejar las necesidades de gestión de los datos de estas aplicaciones hay que dividir los datos entre miles de procesadores.

Durante los últimos años se han desarrollado e implantado un buen número de sistemas para el almacenamiento de datos en la nube que cumplen los requisitos de gestión de información de estas aplicaciones; entre ellos *Bigtable* de Google; *Simple Storage Service (S3)* de Amazon, que proporciona una interfaz web a *Dynamo*, un sistema de almacenamiento clave-valor; *Cassandra*, de Facebook, que es similar a Bigtable y *Sherpa/PNUTS*, de Yahoo!, el componente de almacenamiento de datos del entorno *Azure* de Microsoft, y otros sistemas.

En esta sección se proporciona una descripción general de la arquitectura de estos sistemas de almacenamiento de datos. Aunque algunas personas se refieren a ellos como sistemas de bases de datos distribuidas, no proporcionan muchas de las características que se han visto como norma en los sistemas de bases de datos actuales, como el uso de SQL o las transacciones con propiedades ACID.

19.9.1.1. Representación de datos

Como ejemplo de necesidad de gestión de la información por parte de las aplicaciones web, considere el perfil de un usuario que tiene que estar accesible para un cierto número de aplicaciones que se ejecutan en la organización. El perfil contiene distintos atributos, y se añaden con frecuencia elementos a los atributos del perfil. Algunos atributos pueden contener datos complejos. Normalmente no suele ser suficiente una representación relacional simple para estos datos complejos.

Algunos sistemas de almacenamiento de datos en la nube admiten XML (que se describe en el Capítulo 23) para la representación de estos datos complejos. Otros utilizan la representación **notación de objetos de JavaScript (JSON),** que está teniendo cada vez mayor aceptación para representar datos complejos. Las representaciones en XML y en JSON proporcionan flexibilidad al conjunto de atributos que puede contener un registro, así como al tipo de dichos atributos. Otros como Bigtable definen sus propios modelos de datos para los datos complejos, incluyendo un soporte para registros con un gran número de columnas opcionales. Se volverá a tratar el modelo de datos de Bigtable más adelante en esta sección.

Además, muchas aplicaciones web no necesitan usar un buen lenguaje de consultas o, al menos, pueden utilizarse sin disponer de él. El modo principal de acceso a los datos es guardarlos con una clave asociada y obtener los datos a partir de la clave. En el ejemplo del perfil de usuario anterior, la clave de los datos del perfil de usuario podría ser el identificador del usuario. Existen aplicaciones que conceptualmente necesitan realizar una reunión, pero la implementan como una forma de vista materializada. Por ejemplo, en una aplicación de red social, cada usuario debería ver todos los nuevos mensajes de sus amigos. Desafortunadamente, encontrar el conjunto de amigos y después consultar cada uno sus mensajes puede ocasionar un gran retardo cuando los datos se encuentran distribuidos por un gran número de máquinas. Una alternativa es la siguiente: cuando un usuario envía un mensaje, se envía a todos los amigos de ese usuario, y se actualizan los datos asociados a cada uno de los amigos con un resumen del nuevo mensaje. Cuando el usuario comprueba si hay mensajes nuevos, todos los datos necesarios se encuentran en un mismo lugar y se pueden obtener rápidamente.

Por tanto, los sistemas de almacenamiento en la nube están basados, en su núcleo, en dos funciones primitivas put(clave, valor), para guardar valores con una clave asociada y get(clave), que devuelve el valor guardado asociado a dicha clave. Algunos sistemas como Bigtable proporcionan además consultas sobre intervalos de valores de las claves.

En Bigtable un registro no se guarda como un solo valor, sino que se divide en atributos componentes que se guardan de forma separada. Por tanto, la clave del valor de un atributo conceptualmente consta de identificador de registro y nombre de atributo. Por lo que se refiere a Bigtable, cada valor de atributo es una simple cadena de caracteres. Para obtener todos los atributos de un registro, se usa una consulta sobre un intervalo o, de forma más precisa, una consulta que case con el prefijo formado por el identificador de registro. La función get() devuelve los nombres de los atributos con sus valores. Para obtener de forma eficiente todos los atributos de un registro, el sistema de almacenamiento guarda las entradas ordenadas por las claves, de forma que todos los valores de los atributos de un determinado registro se encuentran agrupados.

De hecho, el identificador de registro se puede estructurar jerárquicamente, aunque para Bigtable el propio identificador de registro es simplemente una cadena de caracteres. Por ejemplo, una aplicación que almacenara las páginas de un robot («araña web») podría asociar una URL como la siguiente:

www.cs.yale.edu/people/silberschatz.html

a un identificador de registro como:

edu.yale.cs.www/people/silberschatz.html

de forma que las páginas quedaran asociadas en grupo en un orden que resultaran útil. Otro ejemplo: el registro de ejemplo de JSON (consulte el ejemplo sobre JSON) se puede representar con un registro de identificador «22222», con varios nombres de atributos como «nombre.nombre», «nombre_departamento», «hijos[1].nombre» o «hijos[2].apellido».

Además, un único ejemplar de Bigtable puede guardar datos de varias aplicaciones, con diversas tablas para cada aplicación, simplemente usando un prefijo con el nombre de la aplicación y el nombre de la tabla como identificador del registro.

Los sistemas de almacenamiento de datos normalmente almacenan varias versiones de los elementos de datos. Las versiones se identifican con una marca de tiempo, pero alternativamente se pueden identificar con un valor entero que se incrementa cuando se crea una nueva versión del dato.

Las búsquedas pueden indicar específicamente la versión del elemento de datos o pueden devolver la versión con mayor número de versión. En Bigtable, por ejemplo, una clave consta de tres partes: (identificador de registro, nombre de atributo, marca de tiempo).

JSON

La notación de objetos de JavaScript, o JSON, es una representación textual de tipos de datos complejos que se usan ampliamente para la transmisión de datos entre aplicaciones, así como para almacenar datos complejos. JSON tiene datos primitivos como enteros, reales y cadenas de caracteres, así como arrays, y «objetos», que son colecciones de pares (nombre de atributo, valor). Este es un ejemplo de objeto JSON:

```
{
  "ID": "22222",
  "nombre": {
    "nombre": "Albert",
    "apellido": "Einstein"
  },
  "nombre_departamento": "Física",
  "hijos": [
    {"nombre": "Hans", "apellido": "Einstein"},
    {"nombre": "Eduard", "apellido": "Einstein"}
  ]
}
```

En el ejemplo anterior se muestra un objeto que contiene pares (nombre de atributo, valor), así como arrays, delimitados por corchetes. JSON se puede ver como una forma simplificada de XML; XML se trata en el Capítulo 23.

Existen bibliotecas para transformar los datos entre la representación en JSON y la representación de objetos que se utiliza en los lenguajes JavaScript y PHP, así como en otros lenguajes de programación.

19.9.1.2. División y recuperación de datos

Dividir los datos es la forma de manejar sistemas de almacenamiento de datos a gran escala. Al contrario que en las bases de datos paralelas habituales, normalmente no es posible elegir una función de división por adelantado. Más aún, si la carga aumenta, se necesitan añadir más servidores y cada servidor debería ser capaz de hacerse cargo de la carga incrementalmente.

Para resolver ambos problemas, los sistemas de almacenamiento de datos normalmente dividen los datos en unidades relativamente pequeñas (pequeñas en estos sistemas significa del orden de cientos de megabytes). Estas divisiones se denominan **tabletas,** reflejando el hecho de que cada tableta es un fragmento de una tabla. La división de los datos se debería hacer sobre la clave de búsqueda, de forma que una solicitud de un valor de una clave se dirija a una única tableta; en otro caso, para cada petición se requeriría el procesamiento en varios sitios, aumentando considerablemente la carga del sistema. Se utilizan dos enfoques: se realiza la división de rango directamente sobre la clave o se lleva a cabo en el resultado de la función de asociación.

El sitio al que se asigna una tableta actúa como sitio principal para dicha tableta. Todas las actualizaciones se dirigen a dicho sitio y las actualizaciones se propagan a las réplicas de la tableta. Las búsquedas también se envían al mismo sitio para que las lecturas sean consistentes con las escrituras.

La división de los datos en tabletas no se fija de antemano, sino que se realiza dinámicamente. Según se insertan los datos, si una tableta crece demasiado, se divide en partes más pequeñas. Incluso si una tableta no es lo bastante grande como para merecer dividirla, si la carga (operaciones *get*/*put*) sobre ella es excesiva, puede ser objeto de división en tabletas más pequeñas, que se puedan distribuir entre dos o más sitios para repartir la carga. Normalmente el número de tabletas es mucho mayor que el número de sitios, por la misma razón que se usa la división virtual en bases de datos paralelas.

Es importante conocer qué sitio de todos es el responsable de una determinada tableta. Se puede hacer teniendo un sitio controlador de tabletas que sigue la función de división, para asociar una petición get() a una o más tabletas y una función de asociación de las tabletas a los sitios, para encontrar qué sitio es el responsable de qué tableta. Cada petición que llega al sistema hay que encaminarla al sitio correcto; si el responsable de esta tarea es un único sitio controlador con una única tableta, enseguida se vería sobrecargado. En su lugar, la información de asociación se puede replicar en un conjunto de sitios encaminadores, que encaminan las peticiones a los sitios con las tabletas apropiadas. Los protocolos de actualización de la información de asociación cuando se divide o se mueve una tableta se diseñan de forma que no se usa bloqueo; una petición podría acabar en un sitio erróneo. El problema consiste en manejar la detección de que el sitio ya no es responsable de la clave indicada en la petición y reencaminar la petición de acuerdo con la información de asociación actualizada.

En la Figura 19.7 se muestra una arquitectura de un sistema de almacenamiento de datos en la nube, basado débilmente en la arquitectura PNUTS. Otros sistemas proporcionan una funcionalidad similar aunque su arquitectura puede variar. Por ejemplo, en Bigtable no existen encaminadores separados; la información de división y de asociación de servidor y tableta se almacena en el sistema de archivos de Google, y los clientes leen la información desde el sistema de archivos y deciden a qué sitio enviar las peticiones.

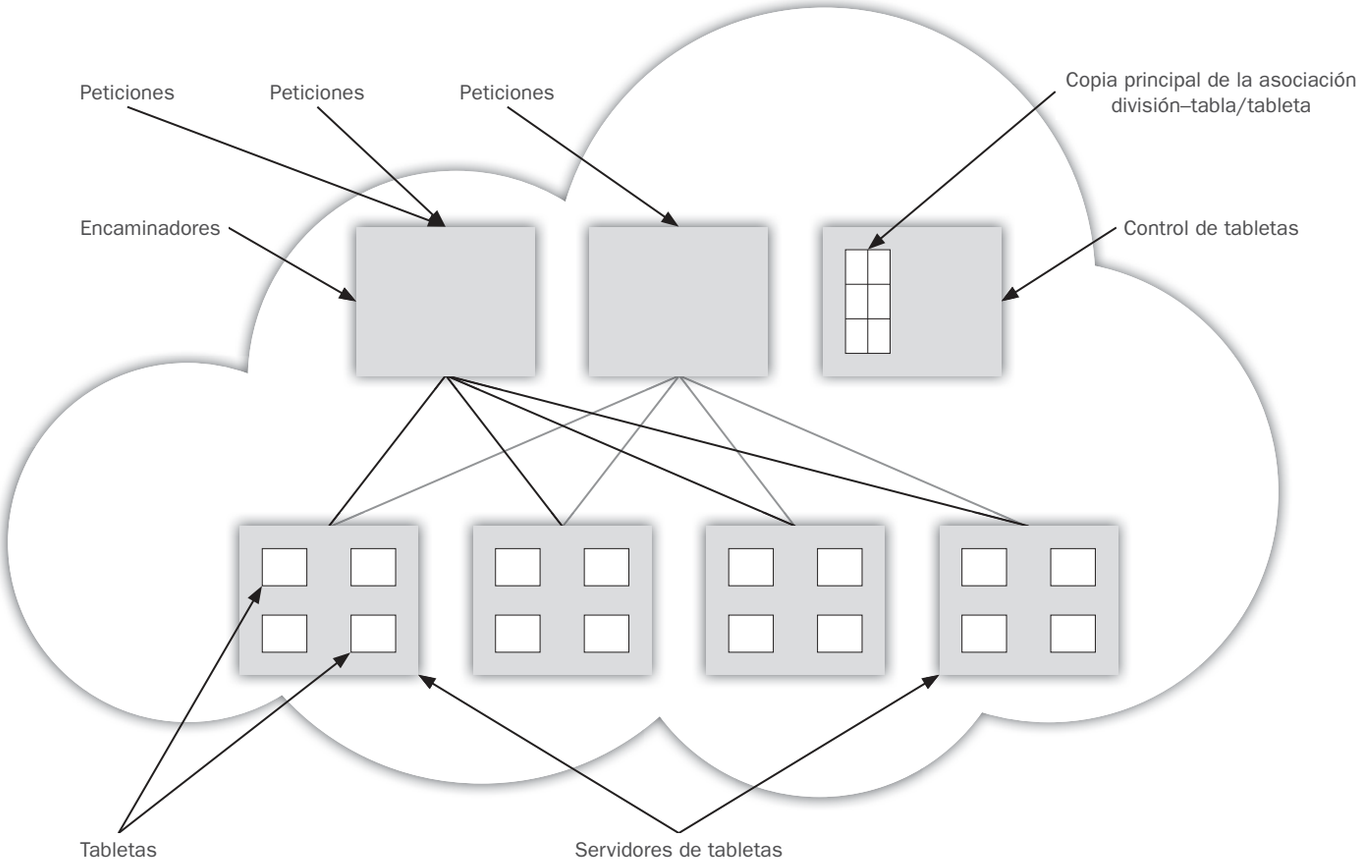


Figura 19.7. Arquitectura de un sistema de almacenamiento de datos en la nube.

19.9.1.3. Transacciones y replicación

Los sistemas de almacenamiento de datos en la nube normalmente no cumplen totalmente con las transacciones ACID. El coste del compromiso de dos fases es demasiado alto y el compromiso de dos fases puede generar el bloqueo en caso de fallos, lo que no es aceptable en las aplicaciones web típicas. Esto significa que estos sistemas normalmente no cumplen con los índices secundarios transaccionalmente consistentes: los índices secundarios serían divididos por un atributo distinto de la clave utilizada para almacenar los datos y una inserción o una actualización requerirían actualizar dos sitios, que necesitarían un compromiso de dos fases. En el mejor de los casos, estos sistemas dispondrían de transacciones en datos para una sola tableta, que la controla un único sitio principal. Sherpa/PNUTS también proporcionan una función *test-and-set*, que permite que una actualización de un elemento de datos sea condicional con el número de versión del elemento de datos que sea igual al número de versión indicado. Si el número de versión actual del elemento de datos es más reciente que el número especificado, la actualización no se realiza. La función *test-and-set* se puede usar en aplicaciones para implementar una forma limitada de control de concurrencia basado en la validación, con la validación restringida a los elementos de datos de una única tableta.

En un sistema con miles de sitios es casi seguro que en un momento dado haya varios sitios que no funcionen. Un sistema de almacenamiento de datos en la nube debe ser capaz de continuar su procesamiento normal incluso con varios sitios fuera de servicio. Estos sistemas replican los datos (como las tabletas) para varias máquinas en una agrupación, de forma que probablemente esté disponible una copia de los datos aunque algunas de las máquinas de la agrupación estén fuera de servicio. (Una **agrupación,** cluster, es una colección de máquinas en un centro de datos). Por ejemplo, el *Sistema de archivos de Google* (GFS–Google File System), que es un sistema de archivos tolerante a fallos distribuidos, replica todos los bloques del sistema de archivos en dos o más nodos de una agrupación. El funcionamiento normal puede continuar siempre que exista al menos una copia de los datos disponible (los datos clave del sistema, como la asociación de archivos a nodos, se replican en más nodos, una mayoría de los cuales deben estar disponibles). Además, la replicación también se usa entre agrupaciones distribuidas geográficamente, por razones que se detallarán un poco más adelante.

Como cada tabla es controlada por un único sitio principal, si el sitio falla la tableta debería reasignarse a un sitio diferente que tenga una copia de la tableta, y de este modo se convierte en el nuevo sitio principal de dicha tableta. Las actualizaciones en la tableta se registran, y el propio registro histórico se replica. Cuando un sitio falla, las tabletas de ese sitio se asignan a otros sitios; el nuevo sitio principal de cada una de las tabletas es responsable de realizar las acciones de recuperación usando el registro histórico para poner la copia de su tableta en un estado consistente y actualizado, tras el cual se pueden realizar actualizaciones y consultas en la tableta.

En Bigtable, por ejemplo, la información de asociación se almacena en una estructura de índices, y estos índices, así como los datos reales de la tableta, se guardan en el sistema de archivos. Las actualizaciones de los datos de la tableta no se escriben inmediatamente, sino solo en el registro histórico. El sistema de archivos asegura que los datos del sistema de archivos están replicados y seguirán estando disponibles aunque se produzca un fallo en algunos pocos nodos de la agrupación. Por tanto, cuando una tableta se reasigna a un nuevo sitio principal tiene acceso a los datos del registro histórico actualizados. El sistema Sherpa/PNUTS de Yahoo!, por otra parte, replica explícitamente las tabletas en nodos de una agrupación, en lugar de usar un sistema de archivos distribuido, y usa un sistema de mensajería distribuido y confiable para implementar un registro histórico de alta disponibilidad.

Por desgracia, no es infrecuente que quede fuera de servicio un centro de proceso de datos completo debido, por ejemplo, a desastres naturales o al fuego. La replicación en un sitio remoto es esencial, por tanto, para disponer de alta disponibilidad. Para muchas aplicaciones web, los retardos de ida y vuelta en una red de gran distancia pueden afectar significativamente al rendimiento, un problema que crece cuando se usan aplicaciones Ajax, que requieren varias idas y vueltas de comunicación entre el navegador y la aplicación. Para combatir este problema, los usuarios se conectan a los servidores de aplicaciones que se encuentren más cercanos geográficamente, y los datos se replican en varios centros de datos de forma que las réplicas se encuentren, muy probablemente, cerca de los servidores de aplicaciones.

Sin embargo, debido a esto, el peligro de división de la red aumenta. Dado que la mayoría de aplicaciones web dan mayor importancia a la disponibilidad que a la consistencia, los sistemas de almacenamiento en la nube normalmente permiten que se realicen las actualizaciones aunque se haya dividido la red, y se proporciona soporte para restaurar la consistencia a posteriori, como se trató antes en la Sección 19.6.6. Para el procesamiento de las actualizaciones suele usarse la replicación multimaestro con propagación perezosa de las actualizaciones, que se vio en la Sección 19.5.3. La propagación perezosa implica que las actualizaciones no se propagan a las réplicas como parte de la transacción de actualización, sino que se propagan en cuanto se pueda, utilizando una infraestructura de mensajería.

Además de propagar las actualizaciones a las réplicas de los elementos de datos, las actualizaciones a los índices secundarios, o a ciertos tipos de vistas materializadas (como las actualizaciones de amigos, en una aplicación de red social que se trató en la Sección 19.9.1.1), se pueden enviar utilizando la infraestructura de mensajería. Los índices secundarios son, básicamente, tablas divididas como tablas normales, según la clave de búsqueda del índice; una actualización de un registro de la tabla se puede asociar a la actualización de una o más tablas en el índice secundario de la tabla. No existe garantía transaccional en las actualizaciones de los índices secundarios ni en las vistas materializadas, solo existe una garantía de máximo esfuerzo en términos de cuándo llegarán a su destino las actualizaciones.

19.9.2. Bases de datos tradicionales en la nube

En este momento se va a abordar la implementación de un sistema distribuido de bases de datos tradicional, con propiedades ACID y consultas, en la nube.

El concepto de utilidades de computación es bastante antiguo, previsto desde los años sesenta. La primera manifestación sobre este concepto se dio en los sistemas de tiempo compartido en los que varios usuarios compartían el acceso a una única computadora mainframe. Más tarde, a finales de los años sesenta, se desarrolló el concepto de **máquinas virtuales,** que ofrecen al usuario la ilusión de disponer de una computadora privada mientras que, en realidad, una única computadora simula varias máquinas virtuales.

La computación en la nube hace un uso extensivo del concepto de máquina virtual para proporcionar servicios de computación. Las máquinas virtuales proporcionan una gran flexibilidad, ya que los clientes pueden elegir su propio entorno de software, no solo el software de aplicación sino también el sistema operativo. Se pueden ejecutar varias máquinas virtuales en una única máquina física si las necesidades de los clientes son bajas. Por otra parte, se puede asignar una computadora en exclusiva a la máquina virtual si un cliente utiliza una máquina virtual con una carga de trabajo muy alta. Un cliente puede solicitar varias máquinas virtuales en las que ejecutar una aplicación. De esta forma resulta sencillo añadir o eliminar potencia de computación, según la carga de trabajo crezca o se reduzca, simplemente añadiendo o eliminando máquinas virtuales.

Disponer de un conjunto de máquinas virtuales funciona bien en las aplicaciones que son fácilmente paralelizables. Los sistemas de bases de datos que se han descrito se encuentran en esta categoría. Cada máquina virtual puede ejecutar el código de un sistema de bases de datos local y comportarse de forma similar a un sitio de un sistema distribuido de bases de datos homogéneas.

19.9.3. Desafíos de las bases de datos en la nube

Las bases de datos en la nube tienen numerosas ventajas frente a construir la infraestructura de computación necesaria desde cero, y de hecho resultan esenciales para ciertas aplicaciones.

Sin embargo, los sistemas basados en la nube también presentan ciertas desventajas que vamos a tratar. Al contrario que en las aplicaciones computacionalmente puras, en las que los cálculos se ejecutan de forma muy independiente, los sistemas distribuidos de bases de datos requieren una frecuente comunicación y coordinación entre sitios para:

- Acceder a los datos de otra máquina física, ya sea porque los datos residen en otra máquina virtual o porque los datos se guardan en un servidor de almacenamiento separado de la computadora en la que reside la máquina virtual.
- Obtener bloqueos en datos remotos.
- Asegurar el compromiso atómico de transacciones mediante el compromiso de dos fases.

En nuestro estudio anterior de las bases de datos distribuidas, se supuso (implícitamente) que el administrador de la base de datos tenía control sobre el sitio físico de los datos. En un sistema en la nube, la ubicación física de los datos se encuentra bajo el control del vendedor, no del cliente. Como consecuencia, la ubicación física de los datos puede ser subóptimo en términos del coste de comunicaciones, y puede suponer un gran número de peticiones de bloqueos remotos y grandes transferencias de datos entre las máquinas virtuales. Para una optimización real de las consultas se requiere que el optimizador tenga medidas precisas del coste de las operaciones. Sin este conocimiento de la ubicación física de los datos, el optimizador tiene que confiar en las estimaciones, que pueden ser muy imprecisas, lo que genera estrategias de ejecución pobres. Como el acceso remoto es relativamente lento comparado con el acceso local, estos temas pueden tener un alto impacto en el rendimiento.

Estos aspectos pueden suponer todo un desafío para la implementación de aplicaciones de bases de datos tradicionales en la nube, aunque menos para los sistemas dedicados solamente al almacenamiento de datos. Los desafíos que se presentan a continuación lo son igualmente para ambos escenarios de aplicaciones.

El tema de la replicación complica más aún la gestión de los datos en la nube. Los sistemas en la nube replican los datos del cliente para aumentar su disponibilidad. De hecho, muchos contratos tienen cláusulas que imponen multas al vendedor si no se cumplen ciertos niveles de disponibilidad. Esta replicación se lleva a cabo por el vendedor sin ningún conocimiento específico de la aplicación. Como la aplicación está bajo el control de la nube y no bajo el control del sistema de bases de datos, se debe poner cuidado cuando el sistema de base de datos accede a los datos para asegurarse que se leen los datos con la última versión. Los fallos a la hora de tener en consideración estos temas puede generar la pérdida de las propiedades de atomicidad o de aislamiento. En muchas aplicaciones de bases de datos en la nube actuales, la propia aplicación puede que tenga que asumir la responsabilidad de la consistencia.

Los usuarios de la computación en la nube deben asumir que los datos se encuentran en otra organización. Esto puede suponer distintos riesgos en términos de seguridad y obligaciones legales. Si el vendedor sufre una brecha de seguridad, los datos del cliente pueden quedar expuestos, lo que provoca que el cliente tenga que hacer frente a denuncias de sus clientes. Aunque el cliente no tenga un control directo sobre la seguridad del vendedor de la nube. Estos temas se vuelven más complejos si el vendedor de la nube elige guardar los datos (o réplicas de los mismos) en otro país. Las normativas legales difieren en sus leyes sobre privacidad. En este sentido, por ejemplo, si los datos de una compañía alemana se replican en un servidor de Nueva York, entonces se aplican las leyes de privacidad de los Estados Unidos, en lugar de las alemanas o las de la Unión Europea. El Gobierno de los Estados Unidos puede requerir al vendedor de la nube que le facilite los datos del cliente incluso aunque este no supiese que sus datos se encontraban bajo la jurisdicción de los Estados Unidos.

Distintos vendedores de sistemas en la nube ofrecen a sus clientes diversos grados de control sobre cómo se distribuyen y replican sus datos. Algunos vendedores ofrecen servicios de bases de datos directamente a sus clientes, en lugar de requerir que los clientes contraten almacenamiento masivo y máquinas virtuales en las que ejecutar sus propias bases de datos.

El mercado de los servicios en la nube continúa evolucionando rápidamente, pero resulta evidente que el administrador de bases de datos que contrata servicios en la nube debe considerar un amplio listado de temas técnicos, económicos y legales para asegurar la privacidad y seguridad de los datos, garantizar las propiedades ACID (o una aproximación aceptable de las mismas) y un rendimiento adecuado a pesar de la probabilidad de que los datos se distribuyan en un área geográfica extensa. En las notas bibliográficas se proporcionan elementos sobre estos temas. Probablemente aparezca mucha nueva literatura en los próximos años, y muchos de los temas actuales sobre bases de datos en la nube se afronten desde la investigación.

19.10. Sistemas de directorio

Considere una organización que desea poner los datos de sus empleados a disposición de diferentes miembros de la organización; entre esos datos estarían el nombre, el cargo, el ID de empleado, la dirección, la dirección de correo electrónico, el número de teléfono, el número de fax, etc. Antes de que existiesen las computadoras, las organizaciones creaban directorios físicos de los empleados y los distribuían por toda la organización. Incluso en nuestros días, las compañías telefónicas crean directorios físicos de sus clientes.

En general, un directorio es un listado de la información sobre alguna clase de objetos como las personas. Los directorios pueden utilizarse para hallar información sobre un objeto concreto o, en sentido contrario, hallar objetos que cumplan un determinado requisito. En el mundo de los directorios telefónicos físicos, los directorios que permiten las búsquedas en sentido directo se denominan **páginas blancas,** mientras que los directorios que permiten las búsquedas en sentido inverso se denominan **páginas amarillas.**

En el mundo interconectado de hoy en día, la necesidad de los directorios sigue vigente y, si acaso, es aún más importante. No obstante, en nuestros días los directorios deben estar disponibles en las redes de computadoras en lugar de en forma física (papel).

19.10.1. Protocolos de acceso a directorios

La información del directorio puede ponerse a disposición de los usuarios mediante interfaces web, como hacen muchas organizaciones y, en especial, las compañías telefónicas. Esas interfaces son buenas para las personas que las utilizan. No obstante, también los programas necesitan tener acceso a la información de los directorios. Estos pueden utilizarse para almacenar otros tipos de información, de manera parecida a como hacen los directorios de sistemas de archivos. Por ejemplo, los exploradores web pueden

almacenar marcas personales de sitios favoritos y otros parámetros del explorador en un sistema de directorio. De esta forma, los usuarios pueden acceder a la misma configuración desde varias ubicaciones, por ejemplo, desde casa y desde el trabajo, sin tener que compartir el sistema de archivos.

Se han desarrollado varios **protocolos de acceso a directorios** para ofrecer una manera normalizada de acceso a los datos contenidos en ellos. Entre todos, el más utilizado hoy en día es el **protocolo ligero de acceso a directorios** (*Lightweight Directory Access Protocol*: **LDAP**).

Evidentemente, todos los tipos de datos de los ejemplos de este capítulo pueden almacenarse sin demasiados problemas en sistemas de bases de datos, y se puede acceder a ellos mediante protocolos como JDBC u ODBC. La pregunta, entonces, es si hace falta crear un protocolo especializado para el acceso a la información del directorio. Al menos hay dos respuestas a esa pregunta:

- En primer lugar, los protocolos de acceso a directorios son protocolos simplificados que atienden a un tipo limitado de acceso a los datos. Han evolucionado en paralelo a los protocolos de acceso a las bases de datos.
- En segundo lugar, y lo que es más importante, los sistemas de directorio ofrecen un mecanismo sencillo para nombrar a los objetos de manera jerárquica, parecida a los nombres de los directorios de los sistemas de archivos, que pueden utilizarse en un sistema distribuido de directorio para especificar la información que se almacena en cada servidor del directorio. Por ejemplo, puede que un servidor de un directorio concreto almacene la información de los empleados de los Laboratorios Bell, en Murray Hill, y que otro almacene la de los empleados de Bangalore, lo que da a ambos sitios autonomía para controlar sus datos locales. Se puede utilizar el protocolo de acceso al directorio para obtener datos de los dos directorios a través de la red, y lo que es más importante, el sistema de directorios puede configurarse para que envíe de manera automática a un sitio las consultas formuladas en el otro, sin intervención del usuario.

Por estos motivos, las organizaciones tienen sistemas de directorios para hacer que la información de la organización esté disponible en conexión mediante un protocolo de acceso al directorio. La información del directorio de la organización se puede utilizar con varios fines, como encontrar la dirección, el teléfono o las direcciones de correo electrónico de la gente, encontrar el departamento al que pertenece cada persona y explorar la jerarquía de los departamentos. Los directorios también se utilizan para autenticar a los usuarios: las aplicaciones pueden recoger la información de autentificación, como las contraseñas de los usuarios, y autenticarla mediante el directorio.

Como cabe esperar, varias implementaciones de los directorios aprovechan las bases de datos relacionales para almacenar los datos, en lugar de crear sistemas de almacenamiento específicos.

19.10.2. El protocolo ligero de acceso a directorios LDAP

En general, los sistemas de directorios se implementan como uno o varios servidores que atienden a diversos clientes. Los clientes utilizan la interfaz de programación de aplicaciones definida por el sistema de directorios para comunicarse con los servidores de directorios. Los protocolos de acceso a los directorios también definen un modelo de datos y el control de los accesos.

El **protocolo de acceso a directorios X.500,** definido por la Organización Internacional para la Normalización (International Organization for Standardization: ISO), es una norma para el acceso a la información de los directorios. No obstante, el protocolo es bastante complejo y no se utiliza mucho. El **protocolo ligero de acceso a directorios (LDAP)** ofrece muchas de las características de X.500, pero con menos complejidad, y se utiliza ampliamente. En el resto de esta sección se esbozarán los detalles del modelo de datos y del protocolo de acceso de LDAP.

19.10.2.1. El modelo de datos LDAP

En LDAP los directorios almacenan **entradas,** que son parecidas a los objetos. Cada entrada debe tener un **nombre distinguido (ND),** que la identifica de manera única. Cada ND, a su vez, está formado por una secuencia de **nombres distinguidos relativos (NDR).** Por ejemplo, una entrada puede tener el siguiente nombre distinguido:

cn=Silberschatz, ou=Informática, o=Yale University, c=USA

Como puede verse, el nombre distinguido de este ejemplo es una combinación de nombre y dirección (organizativa), que comienza por el nombre de la persona y luego da la unidad organizativa (*organizational unit*, ou), la organización (*organization*, o) y el país (*country*, c). El orden de los componentes del nombre distinguido refleja el orden normal de las direcciones postales, en lugar del orden inverso que se utiliza al especificar nombres de camino para los archivos. El conjunto de NDR de cada ND viene definido por el esquema del sistema de directorio.

Las entradas también pueden tener atributos. LDAP ofrece los tipos *binary* (binario), *string* (cadena de caracteres) y *time* (hora) y, de manera adicional, los tipos tel (telefónico) para los números de teléfono y PostalAddress (dirección postal) para las direcciones (las líneas se separan con un carácter «$»). A diferencia de los del modelo relacional, está predeterminado que los atributos puedan tener varios valores, por lo que es posible almacenar varios números de teléfono o direcciones para cada entrada.

LDAP permite la definición de **clases de objetos** con nombres y tipos de atributos. Se puede utilizar la herencia para definir las clases de objetos. Además, se puede especificar que las entradas sean de una clase de objeto o de varias. No es necesario que haya una única clase de objeto más específica a la que pertenezca una entrada dada.

Las entradas se organizan en un **árbol de información del directorio (AID,** o **DIT:** *Directory Information Tree*), de acuerdo con sus nombres distinguidos. Las entradas en el nivel de las hojas del árbol suelen representar objetos concretos. Las entradas que son nodos internos representan objetos como las unidades organizativas, las organizaciones o los países. Los hijos de cada nodo tienen un ND que contiene todos los NDR del padre, y uno o varios NDR adicionales. Por ejemplo, puede que un nodo interno tenga como c=España del ND, y todas las entradas por debajo de él tienen el valor España en el campo c del NDR.

No hace falta almacenar el nombre distinguido completo en las entradas. El sistema puede generar el nombre distinguido de cada una de ellas recorriendo el AID en sentido ascendente desde la entrada, reuniendo los componentes NDR=valor para crear el nombre distinguido completo.

Puede que las entradas tengan más de un nombre distinguido (por ejemplo, la entrada de una persona en más de una organización). Para tratar estos casos, el nivel de las hojas del AID puede ser un **alias** que apunte a una entrada en otra rama del árbol.

19.10.2.2. Tratamiento de los datos

A diferencia de SQL, LDAP no define ni un lenguaje de definición de datos ni uno de tratamiento de datos. Sin embargo, LDAP define un protocolo de red para llevar a cabo la definición y el tratamiento de los datos. Los usuarios de LDAP pueden utilizar tanto una interfaz de programación de aplicaciones como las herramientas ofrecidas por diferentes fabricantes para llevar a cabo la definición y el tratamiento de los datos. LDAP también define un formato de archivos denominado **formato de intercambio de datos LDAP** (*LDAP Data Interchange Format*: *LDIF*) que puede utilizarse para almacenar e intercambiar información.

El mecanismo de consulta en LDAP es muy sencillo, consiste simplemente en selecciones y proyecciones, sin ninguna reunión. Cada consulta debe especificar lo siguiente:

- Una base (es decir, un nodo del AID), dando su nombre distinguido (el camino desde la raíz hasta el nodo).
- Una condición de búsqueda, que puede ser una combinación booleana de condiciones para diferentes atributos. Se soportan la igualdad, la coincidencia con caracteres comodín y la igualdad aproximada (la definición exacta de la igualdad aproximada depende de cada sistema).
- Un ámbito, que puede ser sencillamente la base, la base y sus hijos o todo el subárbol por debajo de la base.
- Los atributos que hay que devolver.
- Los límites impuestos al número de resultados y al consumo de recursos.

La consulta también puede especificar si hay que eliminar de manera automática las referencias de los alias; si se desactiva la eliminación de las referencias de los alias se pueden devolver las entradas de los alias como respuestas.

Una manera de consultar orígenes de datos LDAP es emplear URL de LDAP. Ejemplos de URL de LDAP son:

```
ldap:://codex.cs.yale.edu/o=Yale University,c=USA
ldap:://codex.cs.yale.edu/o=Yale University,c=USA??sub?
        cn=Silberschatz
```

La primera URL devuelve todos los atributos de todas las entradas del servidor en que la organización es Yale University y el país es USA. La segunda URL ejecuta una consulta de búsqueda (selección) cn=Silberschatz en el subárbol del nodo con nombre distinguido o=Yale University, c=USA. Los signos de interrogación de la URL separan los distintos campos. El primer campo es el nombre distinguido, en este caso o=Yale University, c=USA. El segundo campo, la lista de atributos que hay que devolver, se ha dejado vacío, lo que significa que hay que devolver todos los atributos. El tercer atributo, sub, indica que hay que buscar en todo el subárbol. El último parámetro es la condición de búsqueda.

Una segunda manera de consultar los directorios LDAP es utilizar una interfaz de programación de aplicaciones. La Figura 19.8 muestra un fragmento de código C que se utiliza para conectar con un servidor LDAP y ejecutar en él una consulta. El código, en primer lugar, abre una conexión con un servidor LDAP mediante ldap_open y ldap_bind. Luego ejecuta una consulta mediante ldap_search_s. Los argumentos para ldap_search_s son el controlador de conexión LDAP, el ND de la base desde la que se debe realizar la búsqueda, el ámbito de la búsqueda, la condición de búsqueda, la lista de atributos que hay que devolver y un atributo denominado attrsonly que, si se le asigna el valor 1, hace que solo se devuelva el esquema del resultado, sin ninguna tupla real. El último argumento es un argumento de resultados que devuelve el resultado de la búsqueda en forma de estructura LDAPMessage.

El primer bucle **for** itera sobre cada entrada del resultado y la imprime. Observe que cada entrada puede tener varios atributos, y el segundo bucle **for** imprime cada uno de ellos. Dado que los atributos en LDAP pueden tener varios valores, el tercer bucle **for** imprime cada uno de los valores de cada atributo. Las llamadas ldap_msgfree y ldap_value_free liberan la memoria que asignan las bibliotecas LDAP. La Figura 19.8 no muestra el código para el tratamiento de las condiciones de error.

La API[1] de LDAP también contiene funciones para crear, actualizar y borrar entradas, así como para otras operaciones con el AID. Cada llamada a una función se comporta como una transacción independiente; LDAP no soporta la atomicidad de las actualizaciones múltiples.

1 API, interfaz de programación de aplicaciones (del inglés application programming interface). (*N. del E.*)

```
#include <stdio.h>
#include <ldap.h>
main() {
   LDAP *ld;
   LDAPMessage *res, *entry;
   char *dn, *attr, *attrList[] = {"telephoneNumber", NULL};
   BerElement *ptr;
   int vals, i;
   ld = ldap_open("codex.cs.yale.edu", LDAP_PORT);
   ldap_simple_bind(ld, "avi", "avi-passwd");
   ldap_search_s(ld, "o=Yale University, c=USA", LDAP-SCOPE-SUBTREE,
      "cn=Silberschatz", attrList, /*attrsonly*/ 0, &res);
   printf("found %d entries", ldap_count_entries(ld, res));
   for (entry=ldap_first_entry(ld, res); entry != NULL;
      entry = ldap_next_entry(ld, entry))
   {
      dn = ldap_get_dn(ld, entry);
      printf("dn: %s", dn);
      ldap_memfree(dn);
      for (attr = ldap_first_attribute(ld, entry, &ptr);
         attr ! NULL;
         attr = ldap_next_attribute(ld, entry, ptr))
      {
         printf("%s: ", attr);
         vals = ldap_get_values(ld, entry, attr);
         for (i=0; vals[i] != NULL; i++)
            printf("%s, ", vals[i]);
         ldap_value_free(vals);
      }
   }
   ldap_msgfree(res);
   ldap_unbind(ld);
}
```

Figura 19.8. Ejemplo de código LDAP en C.

19.10.2.3. Árboles de directorios distribuidos

La información sobre las organizaciones se puede dividir entre varios AID, cada uno de los cuales almacena información sobre algunas entradas. El **sufijo** de cada AID es una secuencia de pares NDR=valor que identifica la información que almacena ese AID; los pares están concatenados con el resto del nombre distinguido generado al recorrer el árbol desde la entrada hasta la raíz. Por ejemplo, el sufijo de un AID puede ser o=Lucent, c=USA, mientras que otro puede tener el sufijo o=Lucent, c=India. Los AID pueden estar organizativa y geográficamente separados.

Los nodos de un AID pueden contener una **referencia** a un nodo de otro AID; por ejemplo, la unidad organizativa Laboratorios Bell bajo o=Lucent, c=USA puede tener su propio AID, en cuyo caso el AID de o=Lucent, c=USA tendría el nodo ou=LaboratoriosBell, que representará una referencia al AID de Laboratorios Bell.

Las referencias son el componente clave que ayuda a organizar un conjunto distribuido de directorios en un sistema integrado. Cuando un servidor recibe una consulta sobre un AID, puede devolver una referencia al cliente, el cual, a su vez, envía una consulta sobre el AID referenciado. El acceso al AID referenciado es transparente, y se lleva a cabo sin el conocimiento del usuario. Alternativamente, el propio servidor puede formular la consulta al AID referenciado y devolver el resultado junto con el resultado calculado localmente.

El mecanismo jerárquico de nombrado utilizado por LDAP ayuda a repartir el control de la información entre diferentes partes de la organización. La facilidad de las referencias ayuda a integrar todos los directorios de la organización en un solo directorio virtual.

Aunque no sea un requisito LDAP, las organizaciones suelen decidir repartir la información por criterios geográficos (por ejemplo, puede que la organización mantenga un directorio por cada sitio en que tenga una presencia importante) o según su estructura organizativa (por ejemplo, cada unidad organizativa, como puede ser un departamento, mantiene su propio directorio).

Muchas implementaciones de LDAP soportan la réplica maestro-esclavo y la réplica multimaestro de los AID, aunque la réplica no forme parte de la versión actual de la norma LDAP versión 3. El trabajo de normalización de la réplica en LDAP se halla en curso.

19.11. Resumen

- Los sistemas distribuidos de bases de datos constan de un conjunto de sitios, cada uno de los cuales mantiene un sistema local de bases de datos. Cada sitio puede procesar las transacciones locales: las transacciones que solo acceden a datos de ese mismo sitio. Además, cada sitio puede participar en la ejecución de transacciones globales: las que acceden a los datos de varios sitios. La ejecución de las transacciones globales necesita que haya comunicación entre los diferentes sitios.
- Las bases de datos distribuidas pueden ser homogéneas, en las que todos los sitios tienen un esquema y un código de sistemas de bases de datos comunes, o heterogéneas, en las que el esquema y el código de los sistemas pueden ser diferentes.
- Surgen varios problemas relacionados con el almacenamiento de relaciones en bases de datos distribuidas, incluidas la réplica y la fragmentación. Es fundamental que el sistema minimice el grado de conocimiento por los usuarios del modo en que se almacenan las relaciones.
- Los sistemas distribuidos pueden sufrir los mismos tipos de fallos que los sistemas centralizados. No obstante, en los entornos distribuidos hay otros fallos con los que hay que tratar; entre ellos, los fallos de los sitios, los de los enlaces, las pérdidas de mensajes y las divisiones de la red. Es necesario tener en consideración cada uno de esos problemas en el diseño del esquema distribuido de recuperación.
- Para asegurar la atomicidad, todos los sitios en los que se ejecuta la transacción *T* deben estar de acuerdo en el resultado final de su ejecución. O bien *T* se compromete en todos los sitios o se aborta en todos. Para asegurar esta propiedad el coordinador de la transacción de *T* debe ejecutar un protocolo de compromiso. El protocolo de compromiso más empleado es el protocolo de compromiso de dos fases.
- El protocolo de compromiso de dos fases puede provocar bloqueos, situación en la que la terminación de una transacción no se puede determinar hasta que se recupere un sitio que haya fallado (el coordinador). Se puede utilizar el protocolo de compromiso de tres fases para reducir la probabilidad de bloqueo.
- La mensajería persistente ofrece un modelo alternativo para el tratamiento de las transacciones distribuidas. El modelo divide cada transacción en varias partes que se ejecutan en diferentes bases de datos. Se envían mensajes persistentes (que se garantiza que se entregan exactamente una vez, independientemente de los fallos) a los sitios remotos para solicitar que se emprendan acciones en ellos. Aunque la mensajería persistente evita el problema de los bloqueos, los desarrolladores de aplicaciones tienen que escribir código para tratar diferentes tipos de fallos.
- Los diferentes esquemas de control de concurrencia empleados en los sistemas centralizados se pueden modificar para su empleo en entornos distribuidos.
  - En el caso de los protocolos de bloqueo, la única modificación que hay que hacer es el modo en que se implementa el gestor de bloqueos. Existen varios enfoques posibles. Se pueden utilizar uno o varios coordinadores centrales. Si, en vez de eso, se adopta un enfoque con un gestor distribuido de bloqueos, hay que tratar de manera especial los datos replicados.
  - Entre los protocolos para el tratamiento de los datos replicados se hallan el protocolo de copia principal, el de mayoría, el sesgado y el de consenso de quórum. Cada uno de ellos representa diferentes equilibrios en términos de coste y de posibilidad de trabajo en presencia de fallos.
  - En el caso de los esquemas de marcas temporales y de validación, el único cambio necesario es el desarrollo de un mecanismo para la generación de marcas temporales globales únicas.
  - Muchos sistemas de bases de datos soportan la réplica perezosa, en la que las actualizaciones se propagan a las réplicas ubicadas fuera del ámbito de la transacción que ha llevado a cabo la actualización. Estas facilidades deben utilizarse con grandes precauciones, ya que pueden dar lugar a ejecuciones no secuenciables.
- La detección de interbloqueos en entornos con gestor distribuido de bloqueos exige la colaboración entre varios sitios, dado que puede haber interbloqueos globales aunque no haya ningún interbloqueo local.
- Para ofrecer una elevada disponibilidad, las bases de datos distribuidas deben detectar los fallos, reconfigurarse de modo que el cálculo pueda continuar y recuperarse cuando se recupere el procesador o el enlace correspondiente. La tarea se complica enormemente por el hecho de que resulta difícil distinguir entre las divisiones de la red y los fallos de los sitios.

  Se puede extender el protocolo de mayoría mediante el empleo de números de versión para permitir que continúe el procesamiento de las transacciones incluso en presencia de fallos. Aunque el protocolo conlleva una sobrecarga significativa, funciona independientemente del tipo de fallo. Se dispone de protocolos menos costosos para tratar los fallos de los sitios, pero dan por supuesto que no se producen divisiones de la red.
- Algunos algoritmos distribuidos exigen el empleo de coordinadores. Para ofrecer una elevada disponibilidad el sistema debe mantener una copia de seguridad que esté preparada para asumir la responsabilidad si falla el coordinador. Otro enfoque es escoger el nuevo coordinador después de que haya fallado el anterior. Los algoritmos que determinan el sitio que deberá actuar como coordinador se denominan **algoritmos de elección.**
- Puede que las consultas a las bases de datos distribuidas necesiten tener acceso a varios sitios. Se dispone de diversas técnicas de optimización para escoger los sitios a los que hay que tener acceso. Las consultas se pueden reescribir automáticamente en términos de fragmentos de relaciones y las selecciones se pueden realizar en las réplicas de cada fragmento. Se pueden utilizar técnicas de semirreunión para reducir las transferencias de datos de las relaciones en la reunión (o fragmentos o réplicas) entre los distintos sitios.
- Las bases de datos distribuidas heterogéneas permiten que cada sitio tenga su propio esquema y su propio código de sistema de bases de datos. Los sistemas de bases de datos múltiples ofrecen un entorno en el que las nuevas aplicaciones de bases de datos pueden acceder a los datos de gran variedad de bases de datos ya existentes ubicadas en diferentes entornos heterogéneos de hardware y de software. Puede que los sistemas locales de ba-

ses de datos empleen modelos lógicos y lenguajes de definición o de manipulación de datos diferentes y que se diferencien en los mecanismos de control de concurrencia o de administración de las transacciones. Los sistemas de bases de datos múltiples crean la ilusión de la integración lógica de las bases de datos, sin exigir su integración física.

- En los últimos años se han construido un gran número de sistemas de almacenamiento de datos en la nube, en respuesta a las necesidades de almacenamiento a gran escala de las aplicaciones web. Estos sistemas de almacenamiento pueden crecer hasta miles de nodos, con distribución geográfica y alta disponibilidad. Sin embargo, no suelen garantizar las propiedades ACID y consiguen la disponibilidad con divisiones a coste de la consistencia de las réplicas. Los sistemas de almacenamiento actuales no admiten el uso de SQL y solo proporcionan una interfaz simple put()/get(). Aunque la computación en la nube resulta atractiva incluso para las bases de datos tradicionales, existen distintos problemas debidos a la falta de control en la localización de los datos y la replicación geográfica.
- Los sistemas de directorio pueden considerarse una modalidad especializada de base de datos en la que la información se organiza de manera jerárquica, parecida al modo en que los archivos se organizan en los sistemas de archivos. Se accede a los directorios mediante protocolos normalizados de acceso a directorios, como LDAP. Los directorios pueden estar distribuidos entre varios sitios para proporcionar autonomía a cada sitio. Los directorios pueden contener referencias a otros directorios, lo que ayuda a crear vistas integradas en las que cada consulta solo se envía a un directorio y se ejecuta de manera transparente en los directorios correspondientes.

Términos de repaso

- Base de datos distribuida homogénea.
- Base de datos distribuida heterogénea.
- Réplica de datos.
- Copia principal.
- Fragmentación de los datos:
  - Horizontal.
  - Vertical.
- Transparencia de los datos:
  - De la fragmentación.
  - De la réplica.
  - De la ubicación.
- Servidor de nombres.
- Alias.
- Transacciones distribuidas:
  - Locales.
  - Globales.
- Gestor de transacciones.
- Coordinador de transacciones.
- Modos de fallo del sistema.
- División de la red.
- Protocolos de compromiso.
- Protocolo de compromiso de dos fases (C2F):
  - Estado de preparación.
  - Transacciones dudosas.
  - Problema del bloqueo.
- Protocolo de compromiso de tres fases (C3F).
- Mensajería persistente.
- Control de la concurrencia.
- Gestor único de bloqueos.
- Gestor distribuido de bloqueos.
- Protocolos para las réplicas:
  - Copia principal.
  - Protocolo de mayoría.
  - Protocolo sesgado.
  - Protocolo de consenso de quórum.
- Marcas temporales.
- Réplica maestro-esclavo.
- Réplica con varios maestros (actualización distribuida).
- Instantánea consistente con las transacciones.
- Propagación perezosa.
- Tratamiento de los interbloqueos:
  - Grafo local de espera.
  - Grafo global de espera.
  - Ciclos falsos.
- Disponibilidad.
- Robustez:
  - Enfoque basado en la mayoría.
  - Leer uno, escribir todo.
  - Leer uno, escribir todos los disponibles.
  - Reintegración de sitios.
- Selección del coordinador.
- Coordinador suplente.
- Algoritmos de selección.
- Algoritmo de acoso.
- Procesamiento distribuido de consultas.
- Estrategia de semirreunión.
- Sistema de bases de datos múltiples:
  - Autonomía.
  - Mediadores.
  - Transacciones locales.
  - Transacciones globales.
  - Asegurar la secuencialidad global.
  - Billete.
- Computación en la nube.
- Almacenamiento de datos en la nube.
- Tableta.
- Sistemas de directorio.
- Protocolo ligero de acceso a directorios LDAP (*lightweight directory access protocol*):
  - Nombre distinguido (ND).
  - Nombres distinguidos relativos (NDR).
  - Árbol de información del directorio (AID).
- Árboles distribuidos de directorio.
  - Sufijo AID.
  - Referencia.

Ejercicios prácticos

19.1. Indique qué diferencia una base de datos distribuida diseñada para una red de área local de otra diseñada para una red de área amplia.

19.2. Para crear un sistema distribuido con elevada disponibilidad, hay que conocer los tipos de fallos que pueden producirse.

a. Indique los tipos de fallos posibles en los sistemas distribuidos.

b. Indique los elementos de la lista anterior que también sean aplicables a los sistemas centralizados.

19.3. Suponga que se produce un fallo durante la ejecución de C2F para una transacción. Para cada fallo posible de los indicados en el Ejercicio práctico 19.2.a, explique el modo en que C2F garantiza la atomicidad de la transacción a pesar del fallo.

19.4. Suponga un sistema distribuido con dos sitios, *A* y *B*. Indique si el sitio *A* puede distinguir entre:

- *B* deja de funcionar.
- El enlace entre *A* y *B* deja de funcionar.
- *B* está extremadamente sobrecargado y su tiempo de respuesta es cien veces el habitual.

Indique las implicaciones de la respuesta para la recuperación de los sistemas distribuidos.

19.5. El esquema de mensajería persistente descrito en este capítulo depende de las marcas temporales combinadas con el descarte de los mensajes recibidos si son demasiado antiguos. Proponga un esquema alternativo basado en los números de secuencia en lugar de en las marcas temporales.

19.6. Indique un ejemplo en que el enfoque de leer uno, escribir todos los disponibles conduzca a un estado erróneo.

19.7. Explique la diferencia entre la réplica de datos en los sistemas distribuidos y el mantenimiento de sitios de copias de seguridad remotas.

19.8. Indique un ejemplo en el que la réplica perezosa pueda conducir a un estado inconsistente de la base de datos aunque las actualizaciones obtengan un bloqueo exclusivo sobre la copia principal (maestra).

19.9. Considere el siguiente algoritmo de detección de interbloqueos. Cuando la transacción T_i, en el sitio S_1, solicita un recurso a T_j, en el sitio S_3, se envía un mensaje de solicitud con la marca temporal n. Se inserta la arista (T_i, T_j, n) en el grafo local de espera de S_1. La arista (T_i, T_j, n) solo se inserta en el grafo local de espera de S_3 si T_j ha recibido el mensaje de solicitud y no puede conceder de manera inmediata el recurso solicitado. La solicitud de T_i a T_j en el mismo sitio se trata de la manera habitual; no se asocia ninguna marca temporal con el arco (T_i, T_j). El coordinador central invoca el algoritmo de detección enviando un mensaje de inicio a cada sitio del sistema.

Al recibir ese mensaje, cada sitio envía al coordinador su grafo local de espera. Obsérvese que ese grafo contiene toda la información local que tiene cada sitio sobre el estado del grafo real. El grafo de espera refleja un estado instantáneo del sitio, pero no está sincronizado con respecto a ningún otro sitio.

Cuando el controlador ha recibido respuesta de todos los sitios, crea un grafo de la siguiente manera:

- El grafo contiene un vértice para cada transacción del sistema.
- El grafo tiene un arco (T_i, T_j) si y solo si:
  - Existe una arista (T_i, T_j) en uno de los grafos de espera.
  - Aparece un arco (T_i, T_j, n) (para algún n) en más de un grafo de espera.

Demuestre que, si hay un ciclo en el grafo creado, el sistema se halla en estado de interbloqueo y que, si no hay ningún ciclo en el grafo creado, el sistema no se hallaba en estado de interbloqueo cuando comenzó la ejecución del algoritmo.

19.10. Considere una relación que está fragmentada horizontalmente por *número_planta*:

empleado (nombre, dirección, sueldo, número_planta)

Suponga que cada fragmento tiene dos réplicas: una almacenada en el sitio de New York y otra almacenada localmente en el sitio de la planta. Describa una buena estrategia de procesamiento de las consultas siguientes, formuladas en el sitio de San José.

a. Determinar todos los empleados de la planta de Boca.

b. Determinar el sueldo promedio de todos los empleados.

c. Determinar el empleado mejor pagado de cada uno de los sitios siguientes: Toronto, Edmonton, Vancouver y Montreal.

d. Determinar el empleado peor pagado de la compañía.

19.11. Calcule $r \ltimes s$ para las relaciones de la Figura 19.9.

19.12. Indique un ejemplo de una aplicación ideal para la nube y otra que sería difícil de implementar apropiadamente en la nube. Justifique su respuesta.

19.13. Dado que la funcionalidad LDAP puede implementarse sobre un sistema de bases de datos, indique el motivo por el que es necesaria la norma LDAP.

19.14. Considere un sistema de bases de datos múltiples en la que se garantiza que como mucho hay activa una transacción global en un momento dado, y que todos los sitios locales garantizan la secuencialidad.

a. Sugiera formas por las que el sistema de bases de datos múltiples pueda garantizar que como mucho existe una transacción activa en un momento dado.

b. Demuestre con un ejemplo que es posible que exista una planificación global no secuenciable a pesar de estos supuestos.

19.15. Considere un sistema de bases de datos múltiples en el que todos los sitios locales garantizan la secuencialidad local y en el que todas las transacciones globales son de solo lectura.

a. Demuestre con un ejemplo que pueden existir ejecuciones no secuenciables en este tipo de sistema.

b. Demuestre cómo se podría usar un esquema de billetes para asegurar la secuencialidad global.

A	*B*	*C*
1	2	3
4	5	6
1	2	4
5	3	2
8	9	7

r

D	*E*	*F*
3	4	5
3	6	8
2	3	2
1	4	1
1	2	3

s

Figura 19.9. Relaciones para el Ejercicio práctico 19.11.

Ejercicios

19.16. Explique las ventajas relativas de las bases de datos centralizadas y de las distribuidas.

19.17. Explique las diferencias entre transparencia de la fragmentación, transparencia de las réplicas y transparencia de la ubicación.

19.18. Indique cuándo resulta útil tener réplicas de los datos o tenerlos fragmentados. Justifique su respuesta.

19.19. Explique los conceptos de transparencia y de autonomía. Indique el motivo de que estos conceptos sean deseables desde el punto de vista de los factores humanos.

19.20. Si se aplica una versión distribuida del protocolo de granularidad múltiple del Capítulo 15 a una base de datos distribuida, el sitio responsable de la raíz del GAD puede convertirse en un cuello de botella. Suponga que se modifica ese protocolo de la siguiente manera:

- Solo se permiten en la raíz bloqueos en modo tentativo.
- A todas las transacciones se les conceden de manera automática todos los bloqueos posibles en modo tentativo.

Demuestre que estas modificaciones alivian el problema sin permitir planificaciones no secuenciables.

19.21. Estudie y resuma las facilidades que ofrece el sistema de bases de datos que esté utilizando para tratar los estados inconsistentes a los que se puede llegar con la propagación perezosa de las actualizaciones.

19.22. Explique las ventajas e inconvenientes de los dos métodos presentados en la Sección 19.5.2 para la generación de marcas temporales globales únicas.

19.23. Considere las relaciones:

empleado (nombre, dirección, sueldo, número_planta)
máquina (número_máquina, tipo, número_planta)

Suponga que la relación *empleado* está fragmentada horizontalmente por *número_planta* y que cada fragmento se almacena localmente en el sitio de la planta correspondiente. Suponga que toda la relación *máquina* se almacena en el sitio de Armonk. Describa una buena estrategia para el procesamiento de cada una de las consultas siguientes:

a. Determinar todos los empleados de la planta que contiene el número de máquina 1130.

b. Determinar todos los empleados de las plantas que contienen máquinas cuyo tipo sea «trituradora».

c. Determinar todas las máquinas de la planta de Almadén.

d. Determinar *empleado* $\bowtie$ *máquina*.

19.24. Para cada una de las estrategias del Ejercicio 19.23 indique el modo en que la elección de estrategia depende:

a. Del sitio en el que se introduce la consulta.

b. Del sitio en el que se desea obtener el resultado.

19.25. ¿Es necesariamente $r_i \ltimes r_j$ igual a $r_j \ltimes r_i$? ¿En qué circunstancias se cumple que $r_i \ltimes r_j = r_j \ltimes r_i$?

19.26. Si se usa un servicio de almacenamiento en la nube para guardar dos relaciones r y s y se necesita hacer la reunión de r y s, ¿por qué podría ser útil mantener la reunión como una vista materializada? En la respuesta considere distintos significados de «útil»: rendimiento total, eficiencia en el uso del espacio o tiempo de respuesta a las peticiones del usuario.

19.27. ¿Por qué para los servicios de computación en la nube es mejor usar los sistemas de bases de datos tradicionales mediante una máquina virtual en lugar de ejecutarse directamente en la máquina real del proveedor del servicio?

19.28. Describa de qué modo se puede utilizar LDAP para ofrecer varias vistas jerárquicas de los datos sin necesidad de replicar los datos del nivel básico.

Notas bibliográficas

Las bases de datos distribuidas se explican en los libros de texto Ozsu y Valduriez [1999]. Breitbart et ál. [1999b] presentan una visión general de las bases de datos distribuidas.

La implementación del concepto de transacción en las bases de datos distribuidas se presenta en Gray [1981] y Traiger et ál. [1982]. El protocolo C2F lo desarrollaron Lampson y Sturgis [1976]. El protocolo de compromiso de tres fases proviene de Skeen [1981]. Mohan y Lindsay [1983] estudian dos versiones modificadas de C2F, denominadas *presumir compromiso* y *presumir abortar*, que reducen la sobrecarga de C2F mediante la definición de suposiciones predeterminadas relativas a la terminación de las transacciones.

El algoritmo de acoso de la Sección 19.6.5 proviene de García-Molina [1982]. La sincronización distribuida de los relojes se estudia en Lamport [1978]. El control distribuido de la concurrencia se estudia en Bernstein y Goodman [1981].

El gestor de transacciones de R* se describe en Mohan et ál. [1986]. Las técnicas de validación para los esquemas distribuidos de control de concurrencia se describen en Schlageter [1981] y Bassiouni [1988].

El problema de las actualizaciones concurrentes de los datos replicados se revisó en el contexto de los almacenes de datos (*data warehouses*) en Gray et ál. [1996]. Anderson et ál. [1998] estudian problemas relativos a la réplica perezosa y a la consistencia. Breitbart et ál. [1999a] describen los protocolos de actualización perezosa para el tratamiento de réplicas.

Los manuales de usuario de varios sistemas de bases de datos ofrecen detalles del modo en que tratan la réplica y la consistencia. Huang y García-Molina [2001] abordan la semántica de solo-una-vez en los sistemas de mensajería con réplica.

Knapp [1987] estudia la literatura sobre detección distribuida de interbloqueos. El Ejercicio práctico 19.9 procede de Stuart et ál. [1984].

El procesamiento distribuido de las consultas se estudia en Epstein et ál. [1978] y Hevner y Yao [1979]. Daniels et ál. [1982] describen el enfoque del procesamiento distribuido de las consultas adoptado por R*.

La optimización dinámica de las consultas en bases de datos múltiples se aborda en Ozcan et ál. [1997]. Adali et ál. [1996] y Papakonstantinou et ál. [1996] describen los problemas de optimización de las consultas en los sistemas mediadores.

Mehrotra et ál. [2001] tratan el procesamiento de transacciones en sistemas de bases de datos múltiples. El esquema de billetes se presenta en Georgakopoulos et ál. [1994]. 2LSR se presenta en Mehrotra et ál. [1991].

En Ooi y S. Parthasarathy [2009] se presenta una colección de artículos sobre gestión de datos en los sistemas en la nube. La implementación de Bigtable de Google se describe en Chang et ál. [2008], mientras que Cooper et ál. [2008] describen el sistema PNUTS de Yahoo! En Brantner et ál. [2008] se describe la experiencia de construcción de una bases de datos usando el almacenamiento basado en la nube S3 de Amazon. En Lomet et ál. [2009] se trata un enfoque para ejecutar transacciones correctamente en sistemas basados en la nube. El teorema CAP fue una conjetura en Brewer [2000] y fue formalizado y probado por Gilbert y Lynch [2002].

Howes et ál. [1999] tratan en un manual de texto el protocolo LDAP.

Parte 6
Almacenes de datos, minería de datos y recuperación de la información

Las consultas de la base de datos se diseñan normalmente para extraer información específica, como el saldo de una cuenta o la suma de los saldos de la cuenta de un cliente. Sin embargo, las consultas diseñadas para ayudar a formular una estrategia corporativa requieren a menudo acceso a grandes cantidades de datos que provienen de diversas fuentes.

Un almacén de los datos es un repositorio de datos recopilados de múltiples fuentes y almacenados bajo un esquema unificado de la base de datos. Los datos almacenados en almacén se analizan mediante agregaciones complejas y análisis estadísticos, normalmente utilizando SQL para el análisis de los datos como se vio en el Capítulo 5. Además, se pueden utilizar las técnicas de descubrimiento del conocimiento para intentar descubrir reglas y patrones a partir de los datos. Por ejemplo, un minorista puede descubrir que ciertos productos tienden a comprarse juntos, y puede utilizar esa información para desarrollar estrategias de marketing. Este proceso del descubrimiento del conocimiento se llama *minería de datos*. El Capítulo 20 trata estos aspectos de los almacenes de datos y la minería de datos.

En nuestros estudios hasta el momento nos hemos centrado en datos relativamente simples y bien estructurados. Sin embargo, hay una cantidad enorme de datos textuales no estructurados en Internet, intranets en organizaciones y en las computadoras de usuarios individuales. Los usuarios desean encontrar la información relevante de esta gran fuente de información fundamentalmente textual, usando mecanismos simples de consulta como consultas basadas en palabras clave. El campo de la recuperación de información se ocupa de consultar estos datos no estructurados, y presta atención particular a la clasificación de los resultados de la consulta. Aunque este campo de investigación ya tiene varias décadas, ha experimentado un gran crecimiento con el desarrollo de World Wide Web. El Capítulo 21 proporciona una introducción al campo de la recuperación de información.

Almacenes de datos y minería de datos

20

Las empresas han comenzado a aprovechar los cada vez más numerosos datos en línea para tomar mejores decisiones sobre sus actividades, como qué artículos mantener en almacén y cómo dirigirse mejor a los clientes para aumentar las ventas. Existen dos formas de aprovechar estos datos. La primera consiste en recoger datos de múltiples fuentes a un repositorio central, denominado almacén de datos (data warehouse). Entre los aspectos implicados en ello se encuentran las técnicas para tratar con datos «sucios», es decir, datos con errores y con técnicas para el almacenamiento eficiente e indexación de grandes volúmenes de datos.

La segunda consiste en analizar los datos recogidos para descubrir información o conocimiento que pueda sentar las bases para tomar las decisiones de negocio. Algunos tipos de análisis de datos se pueden llevar a cabo usando SQL para el procesamiento analítico en línea (OLAP-online analytical processing), que se vio en la Sección 5.6 (Capítulo 5), y usando herramientas e interfaces gráficas para OLAP. Otro enfoque para obtener información de los datos es utilizar la *minería de datos*, que pretende detectar varios tipos de estructuras en grandes volúmenes de datos. La minería de datos complementa varios tipos de técnicas estadísticas con objetivos parecidos.

20.1. Sistemas de ayuda a la toma de decisiones

Las aplicaciones de bases de datos pueden clasificarse groso modo en sistemas de procesamiento de transacciones y de ayuda a la toma de decisiones. Los sistemas de procesamiento de transacciones son sistemas que registran información sobre las transacciones, como ventas de productos o matrículas e información de titulaciones para las universidades. Los sistemas de procesamiento de transacciones se utilizan mucho hoy en día y las empresas han acumulado una enorme cantidad de información generada por ellos. Los sistemas de ayuda a la toma de decisiones pretenden obtener información de alto nivel a partir de la información detallada que se guarda en los sistemas de transacciones y utilizar esta información de alto nivel para tomar distintas decisiones. Los sistemas de ayuda a la toma de decisiones facilitan a los gestores la decisión sobre los productos de una tienda que debe haber en almacén, los productos que fabricar o las personas a las que se debería admitir en una universidad.

Por ejemplo, las bases de datos de las empresas suelen contener enormes cantidades de información sobre los clientes y las transacciones. La cantidad de información necesaria puede llegar a varios centenares de gigabytes o, incluso, a los terabytes para las cadenas de grandes almacenes. La información de las transacciones de un gran almacén puede incluir el nombre o identificador (como puede ser el número de la tarjeta de crédito) del cliente, los artículos adquiridos, el precio pagado y las fechas en que se realizaron las compras. La información sobre los artículos adquiridos puede incluir el nombre del artículo, el fabricante, el número de modelo, el color y la talla. La información sobre los clientes puede incluir su historial de crédito, sus ingresos anuales, su domicilio, su edad e, incluso, su nivel académico.

Estas bases de datos de gran tamaño pueden resultar minas de información para adoptar decisiones empresariales, como los artículos que debe haber en inventario y los descuentos que hay que ofrecer. Por ejemplo, puede que una cadena de grandes almacenes note un aumento súbito de las compras de camisas de franela en Pacific Northwest, darse cuenta de que hay una tendencia y comenzar a almacenar un mayor número de esas camisas en las tiendas de esa zona. O puede que una empresa automovilística descubra, al consultar su base de datos, que la mayor parte de los coches deportivos de pequeño tamaño los compran mujeres jóvenes cuyos ingresos anuales superan los 50.000 €. Puede que la empresa dirija su publicidad para que atraiga más mujeres de esas características a que compren coches deportivos de pequeño tamaño y evite desperdiciar dinero intentando atraer a otras categorías de consumidores para que compren esos coches. En ambos casos la empresa ha identificado pautas de comportamiento de los consumidores y las ha utilizado para adoptar decisiones empresariales.

El almacenamiento y recuperación de los datos para la ayuda a la toma de decisiones plantea varios problemas:

- Aunque muchas consultas para ayuda a la toma de decisiones pueden escribirse en SQL, otras no pueden expresarse en SQL o no puede hacerse con facilidad. En consecuencia, se han propuesto varias extensiones de SQL para facilitar el análisis de los datos. En la Sección 5.6 se trataron las extensiones de SQL para el análisis de datos y para el *procesamiento analítico en línea* (*online analytical processing*: OLAP).
- Los lenguajes de consulta de bases de datos no son eficaces para el **análisis estadístico** detallado de datos. Existen varios paquetes, como SAS y S++, que ayudan en el análisis estadístico. A estos paquetes se les han añadido interfaces con las bases de datos para permitir que se almacenen en ellas grandes volúmenes de datos y se recuperen de manera eficiente para su análisis. El campo del análisis estadístico es una gran disciplina por sí misma. Consulte las referencias en las notas bibliográficas para obtener más información.
- Las grandes empresas tienen varias fuentes de datos que necesitan utilizar para adoptar decisiones empresariales. Las fuentes pueden almacenar los datos según diferentes esquemas. Por motivos de rendimiento (así como por motivos de control de la organización), las fuentes de datos no suelen permitir que otras partes de la empresa recuperen datos a demanda.

Para ejecutar de manera eficiente las consultas sobre datos tan diferentes las empresas han creado *almacenes de datos* (*data warehouse*). Los almacenes de datos reúnen los datos de varias fuentes bajo un esquema unificado en un solo sitio. Por tanto, ofrecen al usuario una sola interfaz uniforme para los datos. Los problemas de la creación y mantenimiento de los almacenes de datos se estudian en la Sección 20.2.

- Las técnicas de descubrimiento de conocimiento intentan determinar de manera automática reglas estadísticas y patrones a partir de los datos. El campo de la *minería de datos* combina las técnicas de descubrimiento de conocimiento creadas por los investigadores en inteligencia artificial, y los analistas estadísticos con las técnicas de implementación eficiente, que permiten utilizarlas en bases de datos extremadamente grandes. La Sección 20.3 estudia la minería de datos.

El área de **ayuda a la toma de decisiones** puede considerarse que abarca todas las áreas anteriores, aunque algunas personas utilizan el término en un sentido más restrictivo que excluye el análisis estadístico y la minería de datos.

20.2. Almacenes de datos

Las grandes empresas tienen presencia en muchos lugares, cada uno de los cuales puede generar un gran volumen de datos. Por ejemplo, las cadenas de tiendas minoristas poseen centenares o millares de tiendas, mientras que las compañías de seguros pueden tener datos de millares de oficinas locales. Además, las organizaciones grandes tienen una estructura compleja de organización interna y, por tanto, puede que los diferentes datos se hallen en ubicaciones, sistemas operativos o bajo esquemas diferentes. Por ejemplo, puede que los datos de los problemas de fabricación y los datos sobre las quejas de los clientes estén almacenados en diferentes sistemas de bases de datos. Las organizaciones suelen comprar datos de fuentes externas, como listas de correo que se usan para las promociones de los productos, o información de créditos de los clientes que proporcionan las oficinas de crédito sobre la capacidad de estos.[1]

Los encargados de adoptar las decisiones empresariales necesitan tener acceso a la información de todos esos orígenes. La formulación de consultas a cada una de las fuentes es a la vez engorrosa e ineficiente. Además, puede que las fuentes de datos solo almacenen los datos actuales, mientras que es posible que los encargados de adoptar las decisiones empresariales necesiten tener acceso también a datos anteriores, por ejemplo, la información sobre la manera en que se han modificado las pautas de compra el año pasado puede resultar de gran importancia. Los almacenes de datos proporcionan una solución a estos problemas.

Los **almacenes de datos** son repositorios (o archivos) de información reunida de varios orígenes o fuentes, almacenada bajo un esquema unificado en un solo sitio. Una vez reunida la información, los datos se almacenan mucho tiempo, lo que permite el acceso a datos históricos. Así, los almacenes de datos proporcionan a los usuarios una sola interfaz consolidada con los datos, por lo que las consultas de ayuda a la toma de decisiones resultan más fáciles de escribir. Además, al tener acceso a la información para la ayuda de la toma de decisiones desde un almacén de datos, el encargado de adoptar las decisiones se asegura de que los sistemas de procesamiento en línea de las transacciones no se vean afectados por la carga de trabajo de la ayuda de la toma de decisiones.

1 Las oficinas de crédito son compañías que recogen información sobre los consumidores de diferentes fuentes y calculan un baremo sobre la capacidad de cada consumidor.

20.2.1. Componentes de los almacenes de datos

La Figura 20.1 muestra la arquitectura de un almacén de datos típico e ilustra la recogida de los datos, su almacenamiento, así como el soporte de las consultas y del análisis de datos. Entre los problemas que hay que resolver al crear un almacén de datos están los siguientes:

- **Momento y modo de la recogida de los datos.** En una **arquitectura dirigida por el origen** para la recogida de los datos, los orígenes de los datos transmiten la información nueva, bien de manera continua (a medida que se produce el procesamiento de las transacciones), o de manera periódica (de noche, por ejemplo). En una **arquitectura dirigida por el destino,** el almacén de datos envía de manera periódica solicitudes de datos nuevos a los orígenes de datos.

  A menos que las actualizaciones de los orígenes de datos se repliquen en el almacén de datos mediante un compromiso de dos fases, el almacén de datos nunca estará actualizado respecto a los orígenes de datos. El compromiso de dos fases suele resultar demasiado costoso para ser una opción aceptable, por lo que los almacenes de datos suelen tener datos ligeramente desactualizados. Eso, no obstante, no suele suponer un problema para los sistemas de ayuda a la toma de decisiones.
- **Selección del esquema.** Es probable que las fuentes de datos que se han creado de manera independiente tengan esquemas diferentes. De hecho, puede que utilicen diversos modelos de datos. Parte de la labor de los almacenes de datos es llevar a cabo la integración de los esquemas y convertir los datos al esquema integrado antes de almacenarlos. En consecuencia, los datos almacenados en el almacén de datos no son una mera copia de los datos de las fuentes de datos. Por el contrario, se pueden considerar como una vista materializada de los datos de las fuentes de datos.
- **Transformación y limpieza de los datos.** La labor de corregir y realizar un procesamiento previo de los datos se denomina **limpieza de los datos.** Los orígenes de datos suelen entregar datos con numerosas inconsistencias de carácter menor, que pueden corregirse. Por ejemplo, los nombres suelen estar mal escritos y puede que las direcciones tengan mal escritos los nombres de la calle, de la población o de la provincia, o es posible que los códigos postales se hayan introducido de manera incorrecta. Esto puede corregirse en un grado razonable consultando una base de datos de los nombres de las calles y de los códigos postales de cada ciudad. El encaje aproximado de los datos requeridos para esta tarea se conoce como **búsqueda difusa.**

  Las listas de direcciones recogidas de varias fuentes pueden tener duplicados que haya que eliminar en una **operación de mezcla–purga** (esta operación también se conoce como **desduplicación**). Los registros de varias personas de una misma casa pueden agruparse para que solo se realice un envío de correo a cada casa; esta operación se denomina **domiciliación.**

  Los datos se pueden **transformar** de otras formas, además de la limpieza, como cambiar las unidades de medida o convertir los datos a un esquema diferente reuniendo datos de relaciones de varias fuentes. Los almacenes de datos tienen normalmente herramientas gráficas para dar soporte a la transformación de los datos. Estas herramientas permiten que la transformación se especifique con cuadros, y se pueden crear arcos para indicar el flujo de datos. Los cuadros condicionales pueden encaminar datos al siguiente paso apropiado en la transformación. Consulte en la Figura 30.7 un ejemplo de una transformación especificada usando la herramienta gráfica proporcionada por SQL Server de Microsoft.

- **Propagación de las actualizaciones.** Las actualizaciones de las relaciones en los orígenes de datos deben propagarse a los almacenes de datos. Si las relaciones en los almacenes de datos son exactamente las mismas que en las fuentes de datos, la propagación es directa. En caso contrario, el problema de la propagación de las actualizaciones es básicamente un problema de *mantenimiento de las vistas*, que se estudió en la Sección 13.5.
- **Resúmenes de los datos.** Los datos en bruto generados por un sistema de procesamiento de transacciones pueden ser demasiado grandes como para almacenarlos en línea. No obstante, se pueden responder muchas consultas manteniendo únicamente datos-resumen obtenidos por agregación de las relaciones, en lugar de mantener las relaciones enteras. Por ejemplo, en lugar de almacenar los datos de cada venta de ropa, se pueden almacenar las ventas totales de ropa por nombre de artículo y por categoría.

  Suponga que se ha sustituido una relación r por la relación resumen s. Todavía se puede permitir a los usuarios que planteen consultas como si la relación r estuviera disponible en línea. Si la consulta solo necesita datos resumidos, puede que sea posible transformarla en una equivalente utilizando s en lugar de r; consulte la Sección 13.5..

Los distintos pasos implicados para obtener datos a partir de un almacén de datos se denominan **extracción, transformación** y **carga,** o tareas **ETC;** la extracción se refiere a conseguir datos de los orígenes, mientras que la carga se refiere a cargar los datos en el almacén de datos.

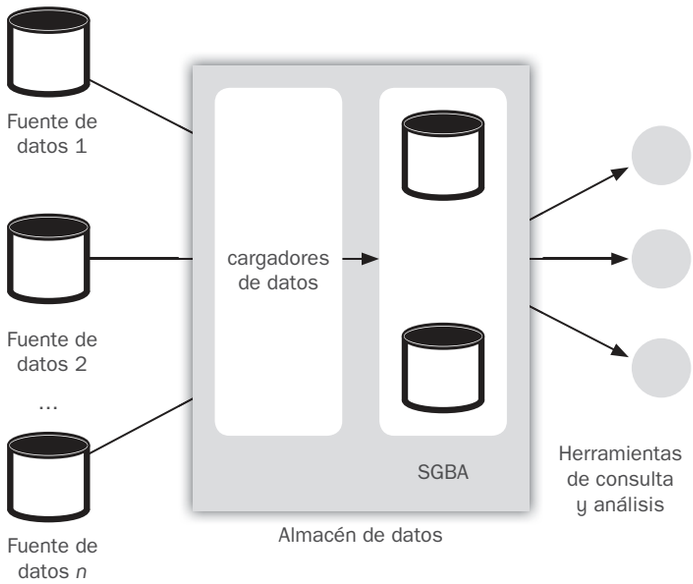


Figura 20.1. Arquitectura de un almacén de datos.

20.2.2. Esquemas de los almacenes de datos

Los almacenes de datos suelen tener esquemas diseñados para el análisis de los datos y emplean herramientas como las herramientas OLAP. Por tanto, los datos suelen ser datos multidimensionales, con atributos de dimensión y atributos de medida. Las tablas que contienen datos multidimensionales se denominan **tablas de hechos** y suelen ser de gran tamaño. Las tablas que registran información de ventas de una tienda minorista, con una tupla para cada artículo a la venta, son un ejemplo típico de tablas de hechos. Las dimensiones de la tabla *ventas* incluyen lo que es el artículo (generalmente un identificador del artículo como el utilizado en los códigos de barras), la fecha en que se ha vendido, la ubicación (tienda) en que se vendió, el cliente que lo ha comprado, etc. Entre los atributos de medida pueden estar el número de artículos vendidos y el precio de cada artículo.

Para minimizar los requisitos de almacenamiento los atributos de dimensiones suelen ser identificadores breves que actúan de claves externas en otras tablas denominadas **tablas de dimensiones.** Por ejemplo, la tabla de hechos *ventas* tendría los atributos *id_artículo*, *id_tienda*, *id_cliente* y *fecha*, así como los atributos de medida *número* y *precio*. El atributo *id_tienda* es una clave externa en la tabla de dimensiones *tienda*, que tiene otros atributos como la ubicación de la tienda (ciudad, provincia, país). El atributo *id_artículo* de la tabla *ventas* es una clave externa de la tabla de dimensiones *info_artículo*, que contiene información como el nombre del artículo, la categoría a la que pertenece el artículo, así como otros detalles de este como el color y la talla. El atributo *id_cliente* es una clave externa de la tabla *cliente*, que contiene atributos como el nombre y la dirección de los clientes. También se puede ver el atributo *fecha* como clave externa de la tabla *info_fecha*, que proporciona el mes, el trimestre y el año de cada fecha.

El esquema resultante aparece en la Figura 20.2. Un esquema así, con una tabla de hechos, varias tablas de dimensiones y claves externas procedentes de la tabla de hechos en las tablas de dimensiones, se denomina **esquema en estrella.** Los diseños complejos de almacenes de datos pueden tener varios niveles de tablas de dimensiones; por ejemplo, la tabla *info_artículo* puede tener un atributo *id_fabricante* que es la clave externa en otra tabla que da detalles del fabricante. Estos esquemas se denominan **esquemas en copo de nieve.** Los diseños complejos de almacenes de datos pueden tener también más de una tabla de hechos.

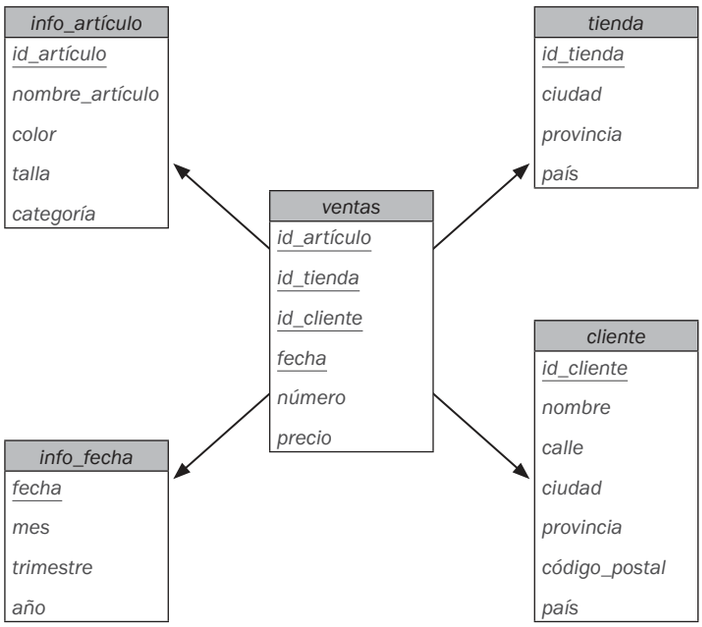


Figura 20.2. Esquema en estrella de un almacén de datos.

20.2.3. Almacenamiento orientado a columnas

Tradicionalmente las bases de datos almacenan todos los atributos juntos en una tupla, y las tuplas se almacenan secuencialmente en filas. Este tipo de almacenamiento se denomina *almacenamiento orientado a filas*. Por el contrario, en un **almacenamiento orientado a columnas,** cada atributo de una relación se guarda en un archivo separado, con los valores de las tuplas sucesivas en posiciones sucesivas del archivo. Suponiendo un tamaño fijo de los datos, el valor del atributo A de la tupla i-ésima de una relación se puede encontrar accediendo al archivo correspondiente al atributo a y leyendo el valor a partir de la posición (i-1) veces el tamaño (en bytes) del valor del atributo.

El almacenamiento orientado a columnas presenta al menos dos ventajas sobre el almacenamiento orientado a filas:

1. Cuando una consulta necesita acceder solo a unos pocos atributos de una relación con un gran número de ellos, el resto de los atributos no hay que obtenerlos de disco a memoria. Por el contrario, en un almacenamiento orientado a filas, no solo se leen en memoria los atributos irrelevantes, sino que se hace una prelectura en la caché del procesador, con la correspondiente pérdida de espacio de caché y de ancho de banda de la memoria, si se almacenan junto a los atributos que se usan en la consulta.
2. Almacenar valores del mismo tipo juntos aumenta la eficacia de la compresión, que puede reducir significativamente tanto el coste de almacenamiento en disco como el tiempo que se tarda en obtener los datos desde el disco.

Por otra parte, el almacenamiento orientado a columnas tiene la desventaja de que guardar u obtener una simple tupla requiere muchas operaciones de E/S.

Como consecuencia de todo ello, el almacenamiento orientado a columna no se usa extensamente para las aplicaciones de procesamiento de transacciones, Sin embargo, está ganando gran aceptación para las aplicaciones de almacenes de datos, en los que los accesos rara vez son a tuplas individuales, sino que se requiere la exploración y agregación de múltiples tuplas.

Sybase IQ fue uno de los primeros productos en usar el almacenamiento orientado a columnas, pero ahora existen muchos proyectos de investigación y compañías que han desarrollado bases de datos basadas en sistemas de almacenamiento orientados a columnas. Estos sistemas han sido capaces de demostrar ganancias de rendimiento significativas en muchas aplicaciones de almacenes de datos. Consulte las notas bibliográficas para más referencias sobre cómo se implementan los almacenes orientados a columnas, y se optimizan y procesan las consultas en estos almacenes.

20.3. Minería de datos

El término **minería de datos** (*data mining*) hace referencia vagamente al proceso de análisis semiautomático de bases de datos de gran tamaño para hallar estructuras útiles. Al igual que la búsqueda de conocimiento en la inteligencia artificial (también denominado aprendizaje automático), o el análisis estadístico, la minería de datos intenta descubrir reglas y estructuras a partir de los datos. No obstante, la minería de datos se diferencia del aprendizaje automático y de la estadística en que trata con grandes volúmenes de datos, almacenados sobre todo en disco. Es decir, la minería de datos trata de la «búsqueda de conocimiento en las bases de datos».

Algunos tipos de conocimiento descubiertos a partir de una base de datos pueden representarse por un conjunto de **reglas.** A continuación se ofrece un ejemplo de regla, formulada de manera informal: «Las mujeres jóvenes con ingresos anuales superiores a 50.000 € son las personas que con mayor probabilidad compran coches deportivos de pequeño tamaño». Por supuesto, estas reglas no son verdaderas de modo universal, y tienen grados de «soporte» y de «confianza» como se estudiará más adelante. Otros tipos de conocimiento se representan mediante ecuaciones que relacionan entre sí diferentes variables, o mediante otros mecanismos de predicción de resultados cuando se conocen los valores de algunas variables.

Hay gran variedad de tipos posibles de estructuras que pueden resultar útiles, y se emplean diversas técnicas para hallar tipos diferentes de estructuras. Se estudiarán unos cuantos ejemplos de estructuras y se verá el modo en que pueden obtenerse de manera automática de las bases de datos.

En la minería de datos suele haber una parte manual, que consiste en el preprocesamiento de los datos hasta una forma aceptable para los algoritmos, y en el posprocesamiento de las estructuras descubiertas para hallar otras nuevas que puedan resultar útiles. También puede haber más de un tipo de estructura que se pueda descubrir a partir de una base de datos dada, y puede que se necesite la interacción manual para escoger los tipos de estructuras útiles. Por este motivo, en la vida real la minería de datos es realmente un proceso semiautomático. No obstante, la descripción que sigue se centrará en el aspecto automático de la minería.

El conocimiento descubierto tiene numerosas aplicaciones. Las aplicaciones más utilizadas son aquellas que requieren algún tipo de **predicción.** Por ejemplo, cuando una persona solicita una tarjeta de crédito, la compañía emisora quiere predecir si la persona tiene un buen riesgo crediticio. La predicción se realiza sobre los atributos de la persona, como edad, ingresos, deudas e historia de pagos de deudas anteriores. Las reglas para realizar la predicción se derivan de los mismos atributos pasados y actuales de los poseedores de tarjetas de crédito, junto con su comportamiento observado, como si dejan de pagar los cargos de las tarjetas. Otros tipos de predicción incluyen predecir qué clientes podrían cambiar a un competidor (a esos clientes se les pueden ofrecer descuentos especiales para tentarles a que no se cambien), predecir qué personas podrían responder a un correo promocional («correo basura»), o predecir qué tipos de uso de tarjetas de teléfono prepago probablemente sean fraudulentas.

Otra clase de aplicaciones buscan **asociaciones,** por ejemplo, los libros que se suelen comprar juntos. Si un cliente compra un libro, la librería en línea puede sugerir otros libros asociados. Si una persona compra una cámara, el sistema puede sugerir accesorios que tiendan a comprarse junto con las cámaras. Un buen vendedor conoce estos patrones y los utiliza para realizar ventas adicionales. El desafío es automatizar el proceso. Otros tipos de asociaciones pueden conducir a descubrir una causalidad. Por ejemplo, descubrir una asociación inesperada entre una nueva medicina y los problemas cardiacos condujo a descubrir que la nueva medicina generaba ciertos problemas cardiacos en algunas personas. La medicina se retiró del mercado.

Las asociaciones son un ejemplo de **patrones descriptivos.** Las **agrupaciones** son otro ejemplo de este tipo de patrones. Por ejemplo, hace aproximadamente un siglo se descubrió un grupo de casos de tifus cerca de un pozo, lo que llevó a descubrir que el agua del pozo estaba contaminada y estaba extendiendo el tifus. La detección de agrupaciones de enfermedades sigue siendo importante en la actualidad.

20.4. Clasificación

Como ya se mencionó en la Sección 20.3, la predicción es uno de los tipos más importantes de minería de datos. Se describirá lo que es la clasificación, se estudiarán técnicas para la creación de un tipo de clasificadores, denominados clasificadores de árboles de decisión, y se estudiarán otras técnicas de predicción.

De manera abstracta, el problema de la **clasificación** es el siguiente: dados elementos que pertenecen a una de varias clases, y dados los casos pasados (denominados **ejemplares de entrenamiento**) de los elementos junto con las clases a las que pertenecen, el problema es predecir la clase a la que pertenece un elemento nuevo. La clase del caso nuevo no se conoce, por lo que hay que utilizar los demás atributos del caso para predecir la clase.

La clasificación se puede llevar a cabo hallando reglas que dividan los datos dados en grupos disjuntos. Por ejemplo, suponga que una compañía de tarjetas de crédito quiera decidir si debe conceder una tarjeta a un solicitante. La compañía tiene amplia información sobre esa persona, como puede ser su edad, su nivel educativo, sus ingresos anuales y sus deudas actuales, la cual puede utilizar para tomar una decisión.

Parte de esta información puede ser importante para el riesgo crediticio del solicitante, mientras que puede que otra parte no lo sea. Para adoptar la decisión, la compañía asigna un nivel de valor de crédito de excelente, bueno, mediano o malo a cada integrante de un conjunto de muestra de clientes *actuales* según su historial de pagos. Luego, la compañía intenta hallar reglas que clasifiquen a sus clientes actuales como excelentes, buenos, medianos o malos con base en la información sobre esas personas diferente a la de su historial de pagos actual (que no está disponible para los clientes nuevos). Considere solo dos atributos: el nivel educativo (la titulación más alta conseguida) y los ingresos. Las reglas pueden ser de la forma siguiente:

$$\forall\ persona\ P,\ P.titulación = máster\ \textbf{and}\ P.ingresos > 75.000 \rightarrow P.crédito = excelente$$

$$\forall\ persona\ P,\ P.titulación = bachiller\ \textbf{or}\ (P.ingresos \geq 25.000\ \textbf{and}\ P.ingresos \leq 75.000) \rightarrow P.crédito = bueno$$

También aparecen reglas parecidas para los demás niveles de riesgo crediticio (mediano y malo).

El proceso de creación de clasificadores comienza con una muestra de los datos, denominada **conjunto de entrenamiento.** Para cada tupla del conjunto de entrenamiento ya se conoce la clase a la que pertenece. Por ejemplo, el conjunto de entrenamiento de las solicitudes de tarjetas de crédito pueden ser los clientes ya existentes, con su riesgo crediticio determinado a partir de su historial de pagos. Los datos actuales, o población, pueden consistir en toda la gente, incluida la que no es todavía cliente. Hay varias maneras de crear clasificadores, como se verá posteriormente.

20.4.1. Clasificadores de árboles de decisión

Los clasificadores de árboles de decisión son una técnica muy utilizada para la clasificación. Como sugiere el nombre, los **clasificadores de árboles de decisión** utilizan un árbol; cada nodo hoja tiene una clase asociada, y cada nodo interno tiene un predicado (o, de manera más general, una función) asociado. La Figura 20.3 muestra un ejemplo de árbol de decisión.

Para clasificar un nuevo caso se empieza por la raíz y se recorre el árbol hasta alcanzar una hoja; en los nodos internos se evalúa el predicado (o la función) para el ejemplar de datos, para hallar a qué nodo hijo hay que ir. El proceso continúa hasta que se llega a un nodo hoja. Por ejemplo, si el nivel académico de la persona es de máster y sus ingresos son de 40K, partiendo de la raíz se sigue la arista etiquetada «máster», y desde allí a la arista etiquetada «25K a 75K», hasta alcanzar una hoja. La clase de la hoja es «bueno», por lo que se puede predecir que la solvencia de esa persona es buena.

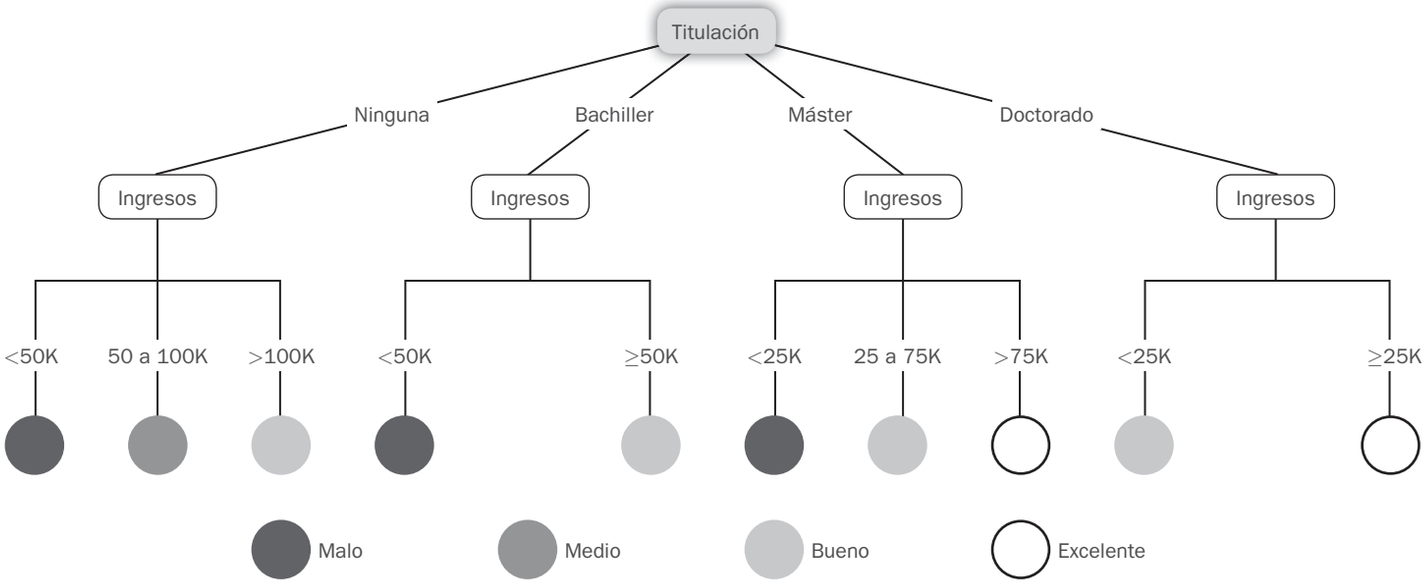


Figura 20.3. Árbol de clasificación.

20.4.1.1. Creación de clasificadores de árboles de decisión

La pregunta que se plantea es el modo de crear un clasificador de árboles de decisión, dado un conjunto de casos de entrenamiento. La manera más frecuente de hacerlo es utilizar un algoritmo **ávido,** que trabaja de manera recursiva, comenzando por la raíz y construyendo el árbol hacia abajo. Inicialmente solo hay un nodo, la raíz, y todos los casos de entrenamiento están asociados a ese nodo.

En cada nodo, si todos o «casi todos» los ejemplares de entrenamiento asociados con el nodo pertenecen a la misma clase, el nodo se convierte en un nodo hoja asociado con esa clase. En caso contrario, hay que seleccionar un **atributo de partición** y **condiciones de partición** para crear nodos hijo. Los datos asociados con cada nodo hijo son el conjunto de ejemplares de entrenamiento que satisfacen la condición de partición de ese nodo hijo. En el ejemplo, se escoge el atributo *titulación* y se crean cuatro hijos, uno por cada valor de la titulación. Las condiciones para los cuatro nodos hijo son *titulación* = ninguna, *titulación* = bachiller, *titulación* = máster y *titulación* = doctorado, respectivamente. Los datos asociados con cada hijo son los ejemplares de entrenamiento asociados con ese hijo. En el nodo correspondiente a máster se escoge el atributo *ingresos* con el rango de valores dividido en los intervalos 0 a 25K, 25K a 50K, 50K a 75K y más de 75K. Los datos asociados con cada nodo son los ejemplares de entrenamiento con atributo *titulación* igual a máster y el atributo *ingresos* en cada uno de los rangos, respectivamente. Como optimización, ya que la clase para el rango de 25K a 50K y el rango de 50K a 75K es el mismo bajo el nodo *titulación* = máster, se han unido los dos rangos en uno solo que va de 25K a 75K.

20.4.1.2. Las mejores particiones

De manera intuitiva, al escoger una secuencia de atributos de partición, se comienza con el conjunto de todos los ejemplares de entrenamiento, que es «impuro», en el sentido de que contiene ejemplares de muchas clases, y se acaba con las hojas, que son «puras», en el sentido de que en cada hoja todos los ejemplares de entrenamiento pertenecen a una única clase. Se verá brevemente el modo de medir cuantitativamente la pureza. Para evaluar la ventaja de escoger un atributo concreto y la condición para la partición de los datos en un nodo se mide la pureza de los datos en los hijos resultantes de la partición según ese atributo. Se escogen el atributo y la condición que producen la pureza máxima.

La pureza de un conjunto S de ejemplares de entrenamiento se puede medir cuantitativamente de varias maneras. Suponga que hay k clases y que de los ejemplares en S la fracción de ejemplares de la clase i es p_i. Una medida de pureza, la **medida de Gini,** se define como:

$$\text{Gini}(S) = 1 - \sum_{i-1}^{k} p_i^2$$

Cuando todos los ejemplares están en una sola clase, el valor de Gini es 0, mientras que alcanza su máximo (de $1-1/k$) si cada clase tiene el mismo número de ejemplares. Otra medida de la pureza es la **medida de la entropía,** que se define como:

$$\text{Entropía(S)} = -\sum_{i-1}^{k} p_i \log_2 p_i$$

la entropía es 0 si todos los ejemplos están en una sola clase y alcanza su máximo cuando cada clase tiene el mismo número de ejemplares. La medida de la entropía proviene de la teoría de la información.

Cuando un conjunto S se divide en varios conjuntos S_i, $i = 1, 2, \ldots, r$, se puede medir la pureza del conjunto de conjuntos resultante como:

$$\text{Pureza}(S_1, S_2, \ldots, S_r) = \sum_{i=1}^{r} \frac{|S_i|}{|S|} \text{pureza}(S_i)$$

Es decir, la pureza es la media ponderada de la pureza de los conjuntos S_i. La fórmula anterior se puede utilizar tanto con la medida de la pureza de Gini como con la medida de la pureza de la entropía.

La **ganancia de información** debida a una partición concreta de S en S_i, $i = 1, 2, \ldots, r$ es, entonces,

$$\text{Ganancia_información}(S, \{S_1, S_2, \ldots, S_r\}) = \text{pureza}(S) - \text{pureza}(S_1, S_2, \ldots, S_r)$$

Las particiones en menor número de conjuntos son preferibles a las particiones en muchos conjuntos, ya que llevan a árboles de decisión más sencillos y significativos. El número de elementos en cada uno de los conjuntos S_i también puede tenerse en cuenta; en caso contrario, que un conjunto S_i tenga uno o ningún elemento supondría una gran diferencia en el número de conjuntos, aunque la partición fuera la misma para casi todos los elementos. El **contenido de información** de una partición concreta puede expresarse en términos de entropía como:

$$\text{Contenido_información}\ (S, \{S_1, S_2, \ldots, S_r\}) = -\sum_{i-1}^{r} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

Todo esto lleva a una definición: la **mejor partición** para un atributo es la que da el máximo **índice de ganancia de información,** definido como:

$$\frac{\text{Ganancia_información}(S, \{S_1, S_2 \ldots, S_r\})}{\text{Contenido_información}(S, \{S_1, S_2 \ldots, S_r\})}$$

20.4.1.3. Búsqueda de las mejores particiones

¿Cómo calcular la mejor partición para un atributo? El modo de dividir un atributo depende del tipo de atributo. Los atributos pueden tener **valores continuos;** es decir, los valores se pueden ordenar de manera significativa para la clasificación, como la edad o los ingresos, o pueden ser **categorías;** es decir, no tener ningún orden significativo, como los nombres de los departamentos o los de los países. No se espera que el orden de los nombres de los departamentos o el de los países tengan ningún significado para la clasificación.

Generalmente, los atributos que son números (enteros o reales) se tratan como valores continuos y los atributos de cadenas de caracteres se tratan como categorías, pero esto puede controlarlo el usuario del sistema. En el ejemplo escogido se ha tratado el atributo *titulación* como categórico y el atributo *ingresos* como valor continuo.

En primer lugar se verá el modo de hallar las mejores particiones para los atributos con valores continuos. Por simplificar solo se considerarán **particiones binarias** de los atributos con valores continuos, es decir, particiones que den lugar a dos hijos. El caso de las **particiones múltiples** es más complicado; consulte las notas bibliográficas para más referencias sobre este tema.

Para hallar la mejor partición binaria de un atributo con valores continuos, en primer lugar se ordenan los valores del atributo en los ejemplares de entrenamiento. Luego se calcula la ganancia de información obtenida por la división en cada valor. Por ejemplo, si los ejemplares de entrenamiento tienen los valores 1, 10, 15 y 25 para un atributo, los puntos de partición considerados son 1, 10 y 15; en cada caso, los valores menores o iguales que el punto de partición forman una partición y el resto de los valores forman la otra. La mejor partición binaria para el atributo es la partición que proporciona la ganancia de información máxima.

Para los atributos categóricos se pueden tener particiones múltiples, con un hijo para cada valor del atributo. Esto funciona muy bien para los atributos categóricos con pocos valores diferentes, como la titulación o el sexo. No obstante, si el atributo tiene muchos valores diferentes, como los nombres de los departamentos en compañías grandes, la creación de un hijo para cada valor no es una buena idea. En esos casos se procura combinar varios valores en cada hijo para crear un número menor de hijos. Consulte las notas bibliográficas para más referencias sobre el modo de hacerlo.

```
procedure CultivarÁrbol(S)
   Partición(S);
procedure Partición(S)
   if (pureza(S) > δp or |S| < δs ) then
      return;
   for each atributo A
      evaluar particiones según el atributo A;
   Usar la mejor partición encontrada (de todos los atributos) para dividir
      S en S1, S2, …, Sr;
   for i = 1, 2, …, r
      Partición(Si );
```

Figura 20.4. Construcción recursiva de un árbol de decisión.

20.4.1.4. Algoritmo de construcción del árbol de decisión

La idea principal de la construcción de árboles de decisión es la evaluación de los diferentes atributos y condiciones de partición y la selección del atributo y de la condición de partición que generen el índice máximo de ganancia de información. El mismo procedimiento funciona de manera recursiva en cada uno de los conjuntos resultantes de la partición, lo que hace que el árbol de decisión se construya de manera recursiva. Si los datos pueden clasificarse de manera perfecta, la recursión se detiene cuando la pureza de un conjunto sea 0. No obstante, los datos suelen tener ruido, o puede que un conjunto sea tan pequeño que no se justifique estadísticamente su partición. En ese caso, la partición se detiene cuando la pureza del conjunto es «bastante alta» y la clase de la hoja resultante se define como la clase de la mayoría de los elementos del conjunto. En general, las diferentes ramas del árbol pueden crecer hasta niveles diferentes.

La Figura 20.4 muestra el seudocódigo para un procedimiento recursivo de construcción de un árbol, que toma al conjunto de ejemplares de entrenamiento S como parámetro. La recursión se detiene cuando el conjunto es lo bastante puro o el conjunto S es demasiado pequeño para que más particiones resulten estadísticamente significativas. Los parámetros δ_p y δ_s definen los valores de corte para la pureza y el tamaño; puede que el sistema tenga valores predeterminados, que los usuarios pueden configurar.

Hay gran variedad de algoritmos de construcción de árboles de decisión y se esbozarán las características distintivas de unos cuantos. Consulte las notas bibliográficas para más detalles. Con conjuntos de datos de tamaño muy grande la realización de particiones puede resultar muy costosa, ya que implica la realización repetida de copias. Por tanto, se han desarrollado varios algoritmos para minimizar el coste de E/S y el coste de computación cuando los datos de entrenamiento son mayores que la memoria disponible.

Varios de los algoritmos también podan los subárboles del árbol de decisión generado para reducir el **exceso de ajuste:** un subárbol tiene exceso de ajuste si se ha ajustado tanto a los detalles de los datos de entrenamiento que comete muchos errores de clasificación con otros datos. Se poda un subárbol sustituyéndolo por un nodo hoja. Hay varias heurísticas de poda; una utiliza parte de los datos de entrenamiento para construir el árbol y otra parte para comprobarlo. La heurística poda el subárbol si descubre que los errores de clasificación de los casos de prueba se reducirían si se sustituyera por un nodo hoja.

Se pueden generar reglas de clasificación a partir de los árboles de decisión, si se desea. Para cada hoja se genera una regla de la siguiente forma: la parte izquierda es la conjunción de todas las condiciones de partición del camino hasta la hoja y la clase es la clase de la mayoría de los ejemplares de entrenamiento de la hoja. Un ejemplo de estas reglas de clasificación es:

$$\textit{titulación} = \textit{máster}\ \textbf{and}\ \textit{ingresos} > 75.000 \rightarrow \textit{excelente}$$

20.4.2. Otros tipos de clasificadores

Hay varios tipos de clasificadores, aparte de los clasificadores de árbol. Han resultado bastante útiles los *clasificadores de redes neuronales*, los *clasificadores bayesianos* y las *máquinas de vectores de soporte*. Los primeros utilizan datos de entrenamiento con el fin de adiestrar redes neuronales artificiales. Hay gran cantidad de literatura sobre redes neuronales y aquí no se hablará más de ellas.

Los **clasificadores bayesianos** hallan la distribución de los valores de los atributos para cada clase de los datos de entrenamiento; cuando se da un nuevo caso, d, utilizan la información de la distribución para estimar, para cada clase c_j, la probabilidad de que el caso d pertenezca a la clase c_j, denotada por $p(c_j|d)$, de la manera que aquí se describe. La clase con la probabilidad máxima se transforma en la clase predicha para el caso d.

Para hallar la probabilidad $p(c_j|d)$ de que el caso d esté en la clase c_j los clasificadores bayesianos utilizan el **teorema de Bayes,** que establece:

$$p(c_j \mid d) = \frac{p(d \mid c_j)p(c_j)}{p(d)}$$

donde $p(d|c_j)$ es la probabilidad de que se genere el caso d dada la clase c_j, $p(c_j)$ es la probabilidad de que se origine la clase c_j, y $p(d)$ es la probabilidad de que ocurra el caso d. De estas, $p(d)$ puede ignorarse, ya que es igual para todas las clases; $p(c_j)$ no es más que la fracción de los casos de entrenamiento que pertenecen a la clase c_j.

Por ejemplo, suponga el caso especial en el que solo se usa un atributo, *ingresos*, para la clasificación y suponga que se necesita clasificar a una persona cuyos ingresos son 76.000 €. Suponga que el valor ingresos se ha dividido en paquetes y que el paquete que contiene el valor 76.000 está en el rango (75.000, 80.000). Suponga que entre los ejemplares de la clase *excelente*, la probabilidad de que los ingresos estén en (75.000, 80.000) es de 0.1, mientras que la probabilidad para la case *bueno* es de 0.05. Suponga, también, que el 0.1 de las personas están clasificadas como *excelentes* y el 0.3 como *bueno*. Entonces, $p(d|c_j)p(c_j)$ para la clase *excelente* es de 0.01, mientras que para la clase bueno es de 0.015. La persona entonces se clasificaría como de la clase *bueno*.

En general hay que considerar varios atributos para la clasificación. Hallar exactamente $p(d|c_j)$ resulta difícil, ya que exige una distribución completa de los casos de c_j en todas las combinaciones de valores para los atributos que se usan para la clasificación. El número de estas combinaciones (por ejemplo de paquetes de ingresos, con valores de titulación y otros atributos) puede ser muy grande. Con un número limitado del conjunto de entrenamiento para encontrar la distribución, la mayoría de las combinaciones no tendrían ni siquiera un caso de entrenamiento que casase, lo que llevaría a decisiones de clasificación erróneas. Para evitar este problema, así como para simplificar la tarea de clasificación, los **clasificadores bayesianos ingenuos** dan por hecho que los atributos tienen distribuciones independientes y, por tanto, estiman:

$$p(d|c_j) = p(d_1|c_j) * p(d_2|c_j) * \ldots * p(d_n|c_j)$$

Es decir, la probabilidad de que ocurra el caso d es el producto de la probabilidad de que ocurra cada uno de los valores d_i del atributo d, dado que la clase es c_j.

Las probabilidades $p(d_i|c_j)$ proceden de la distribución de los valores de cada atributo i, para cada clase c_j. Esta distribución se calcula a partir de los ejemplares de entrenamiento que pertenecen a cada clase c_j; la distribución suele aproximarse mediante un histograma. Por ejemplo, se puede dividir el rango de valores del atributo i en intervalos iguales y almacenar la fracción de casos de la clase c_j que caen en cada intervalo. Dado un valor d_i para el atributo i, el valor de $p(d_i|c_j)$ es simplemente la fracción de casos que pertenecen a la clase c_j que caen en el intervalo al que pertenece d_i.

Una ventaja significativa de los clasificadores bayesianos es que pueden clasificar los casos con valores de los atributos desconocidos y nulos; los atributos desconocidos o nulos simplemente se omiten del cálculo de probabilidades. Por el contrario, los clasificadores de árboles de decisión no pueden tratar de manera significativa las situaciones en las que el caso que hay que clasificar tiene un valor nulo para el atributo de partición utilizado para avanzar por el árbol de decisión.

Las **máquinas de vectores de soporte** (SVM: *support vector machine*) son un tipo de clasificador que se ha visto que hace clasificaciones muy precisas en un gran rango de aplicaciones. Trataremos la intuición básica sobre los clasificadores de máquinas de vectores de soporte; consulte las referencias de las notas bibliográficas para más información.

Los clasificadores de máquinas de vectores de soporte se pueden entender mejor geométricamente. En el caso más simple, suponga un conjunto de puntos en un plano de dos dimensiones, algunos que pertenecen a la clase *A* y otros que pertenecen a la clase *B*. Empezamos con un conjunto de entrenamiento de puntos cuya clase (*A* o *B*) es conocida, y se necesita construir un clasificador de puntos usando estos puntos de entrenamiento. Esta situación se muestra en la Figura 20.5, en la que los puntos de la clase *A* se han marcado con X y los de la clase *B* con O.

Suponga que se dibuja una línea en el plano, de forma que todos los puntos de la clase *A* caen en un lado y todos los puntos de la clase *B* caen del otro lado. Entonces, se puede usar la línea para clasificar nuevos puntos, cuya clase no se conoce todavía. Pero existen muchas posibles líneas que pueden separar los puntos de la clase *A* de los puntos de la clase *B*. Algunas de estas líneas se muestran en la Figura 20.5. El clasificador de la máquina de vectores de soporte elige la línea cuya distancia desde el punto más cercano de cualquiera de las clases (de los puntos del conjunto de datos del entrenamiento) es máxima. Esta línea (llamada *línea de margen máximo*) se usa para clasificar el resto de punto en clase *A* o *B*, dependiendo de en qué lado de la línea caigan. En la Figura 20.5, la línea de margen máximo se muestra en negrita, mientras que el resto de líneas se muestran como líneas de puntos.

La intuición anterior se puede generalizar a más de dos dimensiones, lo que permite usar varios atributos en la clasificación; en este caso, el clasificador encuentra un plano, no una línea. Además, transformando primero los puntos de entrada mediante ciertas funciones, denominadas *funciones kernel*, los clasificadores de máquinas de vectores de soporte pueden encontrar curvas no lineales que separen los conjuntos de puntos. Es importante para los casos en que los puntos no son separables por una línea o un plano. En presencia de ruido, algunos puntos de una de las clases pueden caer en medio de los puntos de la otra clase. En estos casos puede que no exista ninguna línea o curva con sentido que separe los puntos de las dos clases; la línea o curva que separe los puntos de las dos de forma más precisa es la que se elige.

Aunque la formulación básica de las máquinas de vectores de soporte es para clasificadores binarios, es decir, aquellos con dos clases, se puede usar para la clasificación en varias clases de la siguiente forma: si existen *N* clases, se construyen *N* clasificadores, donde el clasificador *i* realiza una clasificación binaria, clasificando un punto como de la clase *i* o no de la clase *i*. Dado un punto, cada clasificador *i* también devuelve un valor indicando cuánto de relacionado está el punto con la clase *i*. Entonces se aplican todos los *N* clasificadores al punto dado, y se elige la clase para la que el valor de relación sea el más alto.

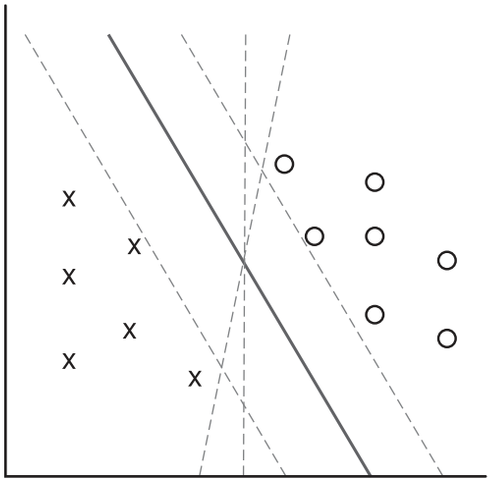


Figura 20.5. Ejemplo de un clasificador de máquina de vectores de soporte.

20.4.3. Regresión

La **regresión** trata de la predicción de valores, no de clases. Dados los valores de un conjunto de variables, $X_1, X_2, \ldots, X_n$, se desea predecir el valor de una variable *Y*. Por ejemplo, se puede tratar el nivel educativo como un número y los ingresos como otro número y, con base en estas dos variables, querer predecir la posibilidad de impago, que podría ser un porcentaje de probabilidad de impago o el importe impagado.

Una manera es inferir los coeficientes $a_0, a_1, a_2, \ldots, a_n$ tales que:

$$Y = a_0 + a_1 * X_1 + a_2 * X_2 + \ldots + a_n * X_n$$

La búsqueda de ese polinomio lineal se denomina **regresión lineal.** En general, se quiere hallar una curva (definida por un polinomio o por otra fórmula) que se ajuste a los datos; el proceso también se denomina **ajuste de la curva.**

El ajuste solo puede ser aproximado, debido al ruido de los datos o a que la relación no sea exactamente un polinomio, por lo que la regresión pretende hallar coeficientes que den el mejor ajuste posible. Hay técnicas estándar en estadística para hallar los coeficientes de regresión. Aquí no se estudiarán esas técnicas, pero las notas bibliográficas ofrecen referencias.

20.4.4. Validación de un clasificador

Es importante validar un clasificador, es decir, medir su tasa de error en la clasificación, antes de decidir utilizarlo en una aplicación. Considere el ejemplo del problema de clasificación en el que un clasificador debe predecir, según ciertas entradas (las entradas exactas no son relevantes), si una persona sufre de una determinada enfermedad *X* o no. Una predicción positiva indica que la persona tiene la enfermedad y una predicción negativa indica que no la tiene. (La terminología de predicción positiva/negativa se puede usar para cualquier problema de clasificación binaria, no solo para la clasificación de enfermedades.)

Se usa un conjunto de casos de prueba en los que el resultado ya se conoce (en el ejemplo, casos en los que ya se sabe si la persona tiene realmente la enfermedad) para medir la calidad (es decir, la tasa de error) del clasificador. Un **verdadero positivo** es el caso en el que la predicción era positiva, y la persona realmente tenía la enfermedad, mientras un **falso positivo** es el caso en el que la predicción era positiva pero la persona no tenía la enfermedad. El **verdadero negativo** y el **falso negativo** se definen de forma similar para casos en que la predicción fue negativa.

Dado un conjunto de casos, sea *v_pos*, *f_pos*, *v_neg* y *f_neg* el número de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. Sea *pos* y *neg* el número real de positivos y negativos (es fácil ver que $pos = v_pos + f_neg$, y que $neg = f_pos + v_neg$).

La calidad de una clasificación se puede medir de varias formas diferentes:

1. **Exactitud,** se define como $(v_pos + v_neg)/(pos + neg)$, es decir, la fracción de las veces que el clasificador da la clasificación correcta.
2. **Memoria** (también llamado **sensibilidad**), se define como v_pos/pos, es decir cuántos de los realmente positivos se clasifican como tales.
3. **Precisión,** se define como $v_pos/(v_pos + f_pos)$, es decir, cómo de frecuente es que la predicción positiva sea correcta.
4. **Especificidad,** definida como v_neg/neg.

Cuáles de estas medidas se pueden utilizar para una aplicación concreta depende de las necesidades de la aplicación. Por ejemplo, es importante una buena memoria para los test de comprobación, que se siguen de pruebas más específicas, de forma que los pacientes con la enfermedad no se confundan. Al contrario, un investiga-

dor que quiera encontrar unos pocos pacientes con la enfermedad a los que seguir pero no esté interesado en todos los pacientes, puede valorar más la precisión que la memoria. Pueden ser apropiados distintos clasificadores dependiendo de las aplicaciones. Este tema se explora más adelante en el Ejercicio 20.5.

Se puede utilizar un conjunto de casos en los que el resultado ya es conocido para el entrenamiento o para medir la calidad del clasificador. Es una mala idea usar exactamente los mismos casos para entrenar y para medir la calidad del clasificador, ya que el clasificador ya ha visto la clasificación correcta de los casos durante el entrenamiento; esto puede llevar a medidas artificiales de calidad. La calidad de un clasificador se debe medir en casos de prueba que no se hayan utilizado durante el entrenamiento.

Por tanto, se usa un subconjunto de los casos de prueba disponibles para el entrenamiento y un subconjunto disjunto para la validación. En la **validación cruzada,** los casos de prueba disponibles se dividen en k partes numeradas de 1 a k, de donde se crean k diferentes conjuntos de prueba de la siguiente forma: el conjunto de prueba i usa la parte i-ésima para la validación, tras el entrenamiento del clasificador con el resto de k-1 partes. Los resultados (*v_pos*, *f_pos*, etc.) de todos los k conjuntos de prueba se añaden antes de calcular las medidas de calidad. La validación cruzada proporciona una medida más precisa de la que se consigue simplemente dividiendo los datos en un único conjunto de entrenamiento y un conjunto de prueba.

20.5. Reglas de asociación

Los comercios minoristas suelen estar interesados en las **asociaciones** entre los diferentes artículos que compra la gente. Ejemplos de esas asociaciones son:

- Alguien que compra pan es bastante probable que compre también leche.
- Una persona que compró el libro *Fundamentos de bases de datos* es bastante probable que también compre el libro *Fundamentos de sistemas operativos*.

La información de asociación puede utilizarse de varias maneras. Cuando un cliente compra un libro determinado puede que la librería en línea le sugiera los libros asociados. Puede que la tienda de alimentación decida colocar el pan cerca de la leche, ya que suelen comprarse juntos, para ayudar a los clientes a hacer la compra más rápidamente. O puede que la tienda los coloque en extremos opuestos del lineal y coloque otros artículos asociados entre medias para inducir a la gente a comprar también esos artículos, mientras los clientes van de un extremo al otro. Puede que una tienda que ofrece descuento en un artículo asociado no lo ofrezca en el otro, ya que de todos modos el cliente comprará el segundo artículo.

Un ejemplo de regla de asociación es:

$$pan \rightarrow leche$$

En el contexto de las compras de alimentación, la regla dice que los clientes que compran pan también tienden a comprar leche con una probabilidad alta. Una regla de asociación debe tener una **población** asociada: la población consiste en un conjunto de **casos.** En el ejemplo de la tienda de alimentación, la población puede consistir en todas las compras en la tienda de alimentación; cada compra es un caso. En el caso de una librería, la población puede consistir en toda la gente que realiza compras, independientemente del momento en que las hayan realizado. Cada consumidor es un caso. Aquí, el analista ha decidido que el momento de realización de la compra no es significativo, mientras que, para el ejemplo de la tienda de alimentación, puede que el analista haya decidido concentrarse en cada compra, ignorando las diferentes visitas de un mismo cliente.

Las reglas tienen un *soporte*, así como una *confianza* asociados. Los dos se definen en el contexto de la población:

- El **soporte** es una medida de la fracción de la población que satisface tanto el antecedente como el consecuente de la regla.

  Por ejemplo, suponga que solo el 0.001 por ciento de todas las compras incluyen leche y destornilladores. El soporte de la regla:

  $$leche \rightarrow destornilladores$$

  es bajo. Puede que la regla ni siquiera sea estadísticamente significativa, quizás solo hubiera una única compra que incluyera leche y destornilladores. Las empresas no suelen estar interesadas en las reglas que tienen un soporte bajo, ya que afectan a pocos clientes y no merece la pena prestarles atención.

  Por otro lado, si el 50 por ciento de las compras implican leche y pan, el soporte de las reglas que afecten al pan y a la leche (y a ningún otro artículo) es relativamente elevado, y puede que merezca la pena prestarles atención. El grado mínimo de soporte que se considera deseable exactamente depende de la aplicación.
- La **confianza** es una medida de la frecuencia con la que el consecuente es cierto cuando lo es el antecedente. Por ejemplo, la regla:

  $$pan \rightarrow leche$$

  tiene una confianza del 80 por ciento si el 80 por ciento de las compras que incluyen pan incluyen también leche. Las reglas con una confianza baja no son significativas. En las aplicaciones comerciales, las reglas suelen tener confianzas significativamente menores del 100 por ciento, mientras que en otros campos, como la física, las reglas pueden tener confianzas elevadas.

  Hay que tener en cuenta que la confianza de $pan \rightarrow leche$ puede ser muy diferente de la confianza de $leche \rightarrow pan$, aunque las dos tienen el mismo soporte.

  Para descubrir reglas de asociación de la forma:

  $$i_1, i_2, \ldots, i_n \rightarrow i_0$$

en primer lugar hay que determinar los conjuntos de elementos con soporte suficiente, denominados **conjuntos grandes de elementos.** En el ejemplo que se trata se hallan conjuntos de elementos que están incluidos en un número de casos lo bastante grande. En breve se verá el modo de calcular conjuntos grandes de elementos.

Para cada conjunto grande de elementos se obtienen todas las reglas con confianza suficiente que afecten a todos los elementos del conjunto y solo a ellos. Para cada conjunto grande de elementos S se obtiene una regla $S - s \rightarrow s$ para cada subconjunto $s \subset S$, siempre que $S - s \rightarrow s$ tenga confianza suficiente; la confianza de la regla la da el soporte de s dividido por el soporte de S.

Ahora se considerará el modo de generar todos los conjuntos grandes de elementos. Si el número de conjuntos de elementos posibles es pequeño, basta con un solo paso por los datos para detectar el nivel de soporte de todos los conjuntos. Se realiza una cuenta, con valor inicial 0, para cada conjunto de elementos. Cuando se obtiene el registro de una compra, la cuenta se incrementa si todos los elementos del conjunto están en la compra. Por ejemplo, si una compra incluye los elementos a, b y c, se incrementará el contador para $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{b, c\}$, $\{a, c\}$ y $\{a, b, c\}$. Los conjuntos con un contador lo bastante elevado al final del pase se corresponden con los elementos que tienen un grado de asociación elevado.

El número de conjuntos crece de manera exponencial, lo que hace inviable el proceso que se acaba de describir si el número de elementos es elevado. Afortunadamente, casi todos los conjuntos tienen normalmente un soporte muy bajo; se han desarrollado optimizaciones para no considerar la mayor parte de esos conjuntos. Estas técnicas utilizan varios pases por la base de datos y solo consideran algunos conjuntos en cada pase.

En la técnica **a priori** para la generación de conjuntos de artículos grandes solo se consideran en el primer pase los conjuntos con un solo elemento. En el segundo pase se consideran los conjuntos con dos artículos, etc.

Al final de cada pase, todos los conjuntos con soporte suficiente se consideran conjuntos grandes de elementos. Los conjuntos que tienen demasiado poco soporte al final de cada pase se eliminan. Una vez eliminado un conjunto no hace falta considerar ninguno de sus superconjuntos. En otros términos, en el pase i solo hay que contar el soporte de los conjuntos de tamaño i tales que se haya hallado que todos sus subconjuntos tienen un soporte lo bastante elevado; basta con probar todos los subconjuntos de tamaño $i-1$ para asegurase de que se cumple esta propiedad. Al final del pase i se halla que ningún conjunto de tamaño i tiene el soporte suficiente, por lo que no hace falta considerar ningún conjunto de tamaño $i+1$. Entonces, el cálculo se termina.

20.6. Otros tipos de asociación

El uso de meras reglas de asociación tiene varios inconvenientes. Uno de los principales es que muchas asociaciones no son muy interesantes, ya que pueden predecirse. Por ejemplo, si mucha gente compra cereales y mucha gente compra pan, se puede predecir que un número bastante grande de personas comprará las dos cosas, aunque no haya ninguna relación entre las dos compras. De hecho, incluso si la compra de cereales tiene una pequeña influencia negativa en la compra de pan (es decir, los clientes que compran cereales tienden a comprar pan menos a menudo que el cliente medio), la asociación entre los cereales y el pan puede tener un soporte alto.

Resultaría interesante que hubiera una **desviación** de la ocurrencia conjunta de las dos compras. En términos estadísticos, se buscan **correlaciones** entre los artículos; las correlaciones pueden ser positivas, cuando la ocurrencia conjunta es superior a lo esperado, o negativa, cuando los elementos ocurren conjuntamente con menos frecuencia de lo predicho. Así, si la compra de pan no se correlaciona con la de cereal, no se informa, aunque hubiese una asociación fuerte entre las dos. Hay medidas estándares de correlación usadas ampliamente en el ámbito de la estadística. Se puede consultar cualquier libro de texto estándar de estadística para obtener más información sobre las correlaciones.

Otra clase importante de aplicaciones de minería de datos son las asociaciones de secuencias (o correlaciones de secuencias). Las series de datos temporales, como las cotizaciones bursátiles en una serie de días, constituyen un ejemplo de datos de secuencias. Los analistas bursátiles desean hallar asociaciones entre las secuencias de cotizaciones. Un ejemplo de asociación de este tipo es la regla siguiente: «Siempre que las tasas de interés de los bonos suben, las cotizaciones bursátiles bajan en un plazo de dos días». El descubrimiento de esta asociación entre secuencias puede ayudar a adoptar decisiones de inversión inteligentes. Consulte las notas bibliográficas para más información sobre la investigación en este campo.

Las desviaciones de las estructuras temporales suelen resultar interesantes. Por ejemplo, si una empresa ha estado creciendo a una tasa constante cada año, una desviación de la tasa de crecimiento habitual resulta sorprendente. Si las ventas de ropa de invierno bajan en verano, ya que puede predecirse con base en los años anteriores, una desviación que no se pudiera predecir a partir de la experiencia pasada se consideraría interesante. Las técnicas de minería pueden hallar desviaciones de lo esperado con base en las estructuras temporales o secuenciales pasadas. Consulte las notas bibliográficas para más información sobre la investigación en este campo.

20.7. Agrupamiento

De manera intuitiva, el agrupamiento hace referencia al problema de hallar agrupaciones de puntos en los datos dados. El problema del **agrupamiento** puede formalizarse de varias maneras a partir de las métricas de distancias. Una manera es formularlo como el problema de agrupar los puntos en k conjuntos (para un k dado) de modo que la distancia media de los puntos al *centroide* de su agrupación asignada sea mínima.[2] Otra manera es agrupar los puntos de modo que la distancia media entre cada par de puntos de cada agrupación sea mínima. Hay otras definiciones; consulte las notas bibliográficas para más información. Pero la intuición subyacente a todas estas definiciones es agrupar los puntos parecidos en un único conjunto.

Otro tipo de agrupamiento aparece en los sistemas de clasificaciones de la biología (estos sistemas de clasificación no intentan *predecir* las clases, sino agrupar los elementos relacionados). Por ejemplo, los leopardos y los seres humanos se agrupan bajo la clase mamíferos, mientras que los cocodrilos y las serpientes se agrupan bajo la de reptiles. Tanto los mamíferos como los reptiles están bajo la clase común de los cordados. La agrupación de los mamíferos tiene subagrupaciones, como los carnívoros y los primates. Por tanto, se tiene un **agrupamiento jerárquico.** Dadas las características de las diferentes especies, los biólogos han creado un esquema complejo de agrupamiento jerárquico que agrupa las especies relacionadas en diferentes niveles de la jerarquía.

El agrupamiento jerárquico también resulta útil en otros dominios; para agrupar documentos, por ejemplo. Los sistemas de directorio de Internet (como el de Yahoo!) agrupan los documentos relacionados de manera jerárquica (consulte la Sección 21.9). Los algoritmos de agrupamiento jerárquico pueden clasificarse como algoritmos de **agrupamiento aglomerativo,** que comienzan creando agrupaciones pequeñas y luego crean los niveles superiores, o como algoritmos de **agrupamiento divisivo,** que primero crean los niveles superiores del agrupamiento jerárquico y luego refinan cada agrupación resultante en agrupaciones de niveles inferiores.

La comunidad estadística ha estudiado extensamente los agrupamientos. La investigación en bases de datos ha proporcionado algoritmos escalables de agrupamiento que pueden agrupar conjuntos de datos de tamaño muy grande (que puede que no quepan en la memoria). El algoritmo de agrupamiento Birch es un algoritmo de este tipo. De manera intuitiva, los puntos de datos se insertan en una estructura de árbol multidimensional (basada en los árboles R descritos en la Sección 25.3.5.3) y son llevados a los nodos hoja correspondientes de acuerdo con su cercanía a los puntos representativos de los nodos internos del árbol. Los puntos próximos, por tanto, se agrupan en los nodos hoja, y se resumen si hay más puntos de los que caben en la memoria. El resultado de esta primera fase de agrupamiento es crear un conjunto de datos agrupado parcialmente que quepa en la memoria. Las técnicas estándares de agrupamiento se pueden ejecutar con datos en la memoria para conseguir el agrupamiento final. Consulte las notas bibliográficas para más información sobre el algoritmo Birch y otras técnicas de agrupamiento, incluidos los algoritmos para el agrupamiento jerárquico.

2 El centroide de un conjunto de puntos se define como un punto cuyas coordenadas en cada dimensión son el promedio de las coordenadas de todos los puntos de ese conjunto en esa dimensión. Por ejemplo, en dos dimensiones, el centroide de un conjunto de puntos $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ viene dado por $\left(\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n}\right)$.

Una aplicación interesante del agrupamiento es la predicción de las películas nuevas (o de los libros nuevos, o de la música nueva) que es probable que interesen a una persona, con base en:

1. Las preferencias cinematográficas pasadas de esa persona.
2. Otras personas con preferencias pasadas parecidas.
3. Las preferencias de esas personas por películas nuevas.

Un enfoque de este problema es el siguiente. Para hallar gente con preferencias anteriores parecidas se crean agrupaciones de personas de acuerdo con sus preferencias cinematográficas. La exactitud del agrupamiento puede mejorarse agrupando previamente las películas por su parecido, de modo que, aunque la gente no haya visto las mismas películas, se agruparán si han visto películas parecidas. Se puede repetir el agrupamiento agrupando alternativamente gente y películas hasta que se alcance un equilibrio.

Dado un nuevo usuario, se halla una agrupación de usuarios lo más parecidos que sea posible a él, sobre la base de las preferencias del usuario por las películas que ya ha visto. Luego se predice qué películas de las agrupaciones de películas que son populares en la agrupación de ese usuario es probable que resulten interesantes para el nuevo usuario. De hecho, este problema es un caso de *filtrado colaborativo*, en el que los usuarios colaboran en la tarea de filtrado de la información para hallar información de interés.

20.8. Otros tipos de minería

La **minería de textos** aplica las técnicas de minería de datos en documentos de texto. Por ejemplo, hay herramientas que forman agrupaciones de las páginas que ha visitado un usuario; esto ayuda a los usuarios cuando examinan su historial de exploración para hallar las páginas que han visitado anteriormente.

La distancia entre las páginas puede basarse, por ejemplo, en las palabras frecuentes en esas páginas (consulte la Sección 21.2.2). Otra aplicación es la clasificación automática de las páginas en directorios web, de acuerdo con su parecido con otras páginas (consulte la Sección 21.9).

Los sistemas de **visualización de datos** ayudan a los usuarios a examinar grandes volúmenes de datos y a detectar visualmente las estructuras. Las visualizaciones de datos —como los mapas, los gráficos y otras representaciones gráficas— permiten que los datos se presenten a los usuarios de manera compacta.

Una sola pantalla gráfica puede codificar tanta información como un número mucho mayor de pantallas de texto. Por ejemplo, si el usuario desea averiguar si los problemas de producción en las factorías están correlacionados con su ubicación se pueden codificar las ubicaciones problemáticas con un color especial —por ejemplo, rojo— en un mapa. El usuario puede descubrir rápidamente las ubicaciones en las que se dan los problemas. El usuario puede así formular hipótesis sobre el motivo que hace que los problemas se produzcan en esas ubicaciones y contrastarlas cuantitativamente con la base de datos.

Otro ejemplo más: la información sobre los valores puede codificarse con colores y mostrarse con solo un píxel de área de pantalla. Para detectar las asociaciones entre pares de elementos se puede utilizar una matriz bidimensional de píxeles en la que cada fila y cada columna representen un elemento. El porcentaje de transacciones que compran los dos elementos puede codificarse por la intensidad del color de los píxeles. Los elementos con asociaciones elevadas aparecerán en la pantalla como píxeles brillantes; fáciles de detectar contra el fondo más oscuro.

Los sistemas de visualización de datos no detectan de manera automática las estructuras, sino que proporcionan soporte del sistema para que los usuarios las detecten. Dado que los seres humanos son muy buenos en la detección de estructuras visuales, la visualización de los datos es un componente importante de la minería de datos.

20.9. Resumen

- Los sistemas de ayuda a la toma de decisiones analizan en línea los datos recogidos por los sistemas de procesamiento de transacciones para ayudar a los usuarios a adoptar decisiones de negocios. Dado que en la actualidad la mayoría de las organizaciones están intensamente informatizadas, se dispone de una enorme cantidad de información para la ayuda a la toma de decisiones. Los sistemas de ayuda a la toma de decisiones se presentan en varios formatos, incluidos los sistemas OLAP y los sistemas de minería de datos.
- Los almacenes de datos ayudan a obtener y archivar datos operacionales importantes. Los almacenes se usan para la ayuda a la toma de decisiones y para el análisis de datos históricos, por ejemplo, para predecir tendencias. La limpieza de datos de orígenes de datos de entrada es a menudo una tarea importante en los almacenes de datos. Los esquemas de los almacenes de datos tienden a ser multidimensionales, implicando una o varias tablas muy grandes de hechos y varias tablas de dimensiones mucho más pequeñas.
- El almacenamiento orientado a columnas proporciona un buen rendimiento para muchas aplicaciones de almacén de datos.
- La minería de datos es el proceso de análisis semiautomático de bases de datos de gran tamaño para hallar estructuras útiles. Hay gran número de aplicaciones de minería de datos, como la predicción de valores con base en los ejemplos ya pasados, la búsqueda de asociaciones entre las compras, así como la agrupación automática de personas y películas.
- La clasificación aborda la predicción de la clase de los ejemplos de prueba utilizando los atributos de los ejemplares de prueba sobre la base de los atributos de los ejemplares de entrenamiento y la clase real de los ejemplares de entrenamiento. La clasificación puede utilizarse, por ejemplo, para predecir el riesgo crediticio de los nuevos solicitantes o para predecir el rendimiento de los estudiantes que solicitan el ingreso en una universidad. Hay varios tipos de clasificadores:
  - Los clasificadores de árboles de decisión llevan a cabo la clasificación creando un árbol basado en los ejemplares de entrenamiento con hojas que tienen las etiquetas de las clases. Se recorre el árbol para cada ejemplo de prueba hasta hallar una hoja y la clase de esa hoja es la clase predicha. Se dispone de varias técnicas para crear árboles de decisión, la mayor parte de ellas basadas en heurísticas ávidas.
  - Los clasificadores bayesianos son más sencillos de crear que los clasificadores de árboles de decisión y funcionan mejor en caso de que haya valores de los atributos que falten o que sean nulos.
  - Las máquinas de vectores de soporte es otra técnica de clasificación ampliamente utilizada.
- Las reglas de asociación identifican los elementos que aparecen juntos con frecuencia, por ejemplo, los artículos que el mismo cliente suele comprar. Las correlaciones buscan las desviaciones respecto de los niveles de asociación esperados.
- Otros tipos de minería de datos son el agrupamiento, la minería de textos y la visualización de datos.

Términos de repaso

- Sistemas de ayuda a la toma de decisiones.
- Análisis estadístico.
- Almacenes de datos.
  - Recogida de datos.
  - Arquitectura dirigida por el origen.
  - Arquitectura dirigida por el destino.
  - Limpieza de datos.
    - Mezcla-purga.
    - Domiciliación.
  - Extracción, transformación y carga (ETC).
- Esquemas de almacenamiento.
  - Tabla de hechos.
  - Tablas de dimensiones.
  - Esquema en estrella.
- Almacenamiento orientado a columnas.
- Minería de datos.
- Predicción.
- Asociaciones.
- Clasificación.
  - Datos de entrenamiento.
  - Datos de prueba.
- Clasificadores de árboles de decisión.
  - Atributo de partición.
  - Condición de partición.
  - Pureza.
    - Medida de Gini.
    - Medida de la entropía.
  - Ganancia de información.
  - Contenido de la información.
  - Tasa de ganancia de la información.
  - Atributo con valores continuos.
  - Atributo categórico.
  - División binaria.
  - División múltiple.
  - Exceso de ajuste.
- Clasificadores bayesianos.
  - Teorema de Bayes.
  - Clasificadores bayesianos ingenuos.
- Máquina de soporte de vectores (MSV).
- Regresión.
  - Regresión lineal.
  - Ajuste de curvas.
- Validación.
  - Exactitud.
  - Memoria (sensibilidad).
  - Precisión.
  - Especificidad.
  - Validación cruzada.
- Reglas de asociación.
  - Población.
  - Soporte.
  - Confianza.
  - Conjuntos de elementos de gran tamaño.
- Otros tipos de asociación.
- Agrupamiento.
  - Jerárquico.
  - Aglomerativo.
  - Divisivo.
- Minería de textos.
- Visualización de datos.

Ejercicios prácticos

20.1. Describa las ventajas e inconvenientes de una arquitectura dirigida por el origen para la recolección de datos en los almacenes de datos en comparación con una arquitectura dirigida por el destino.

20.2. Indique por qué el almacenamiento orientado a columnas puede tener ventajas en un sistema de bases de datos que soporta un almacén de datos (data warehouse).

20.3. Suponga que hay dos reglas de clasificación, una que dice que las personas con sueldos entre 10.000 € y 20.000 € tienen una calificación de crédito de *buena*, y otra que dice que las personas con sueldos entre 20.000 € y 30.000 € tienen una calificación de crédito de *buena*. Indique las condiciones para las que se pueden reemplazar ambas reglas, sin pérdida de información, por una sola regla que diga que las personas con sueldos entre 10.000 € y 30.000 € tienen una calificación de crédito de *buena*.

20.4. Considere el esquema de la Figura 20.2. Escriba una consulta de SQL para resumir las cifras de ventas y los precios por tienda y por fecha, junto con las jerarquías para tienda y fecha.

20.5. Suponga un problema de clasificación en el que el clasificador predice si una persona tiene una determinada enfermedad. Suponga que el 95% de las personas probadas no sufren la enfermedad, es decir, *pos* es el 5% y *neg* es el 95% de los casos de prueba. Considere los siguientes clasificadores:

- Clasificador C_1 que siempre predice negativo (un clasificador poco útil).
- Clasificador C_2 que predice positivo un 80% de los casos en que la persona realmente tiene la enfermedad, pero también predice positivo en un 5% de los casos en que la persona no tiene la enfermedad.
- Clasificador C_3 que predice positivo en el 95% de los casos en que la persona realmente tiene la enfermedad, pero también predice positivo en el 20% de los casos en que la persona no tiene la enfermedad.

Dados los clasificadores anteriores, responda a las siguientes preguntas:

a. Para cada uno de los clasificadores anteriores, calcule exactitud, precisión, sensibilidad y especificidad.

b. Si se desea usar los resultados de clasificación para realizar más pruebas de la enfermedad, ¿cómo seleccionaría el clasificador?

c. Por otra parte, si se pretende usar los resultados de la clasificación para empezar la medicación, que puede tener graves efectos secundarios si se da a alguien que no tiene la enfermedad, ¿cómo seleccionaría los clasificadores?

Ejercicios

20.6. Dibuje un diagrama que muestre cómo se almacenaría en una estructura orientada a columna la relación *aula* del ejemplo de la universidad que puede encontrar en el Apéndice A.

20.7. Explique por qué el algoritmo de reunión de bucle anidado (consulte la Sección 12.5.1) se comportaría de forma pobre en bases de datos con almacenamiento orientado a columna. Describa un algoritmo alternativo que funcionaría mejor y explique por qué su solución es mejor.

20.8. Construya un clasificador de árboles de decisión con divisiones binarias en cada nodo utilizando las tuplas de la relación $r(A, B, C)$ que se muestran a continuación como datos de entrenamiento; el atributo C indica la clase. Muestre el árbol final y, con cada nodo, muestre la mejor división para cada atributo junto con su valor de ganancia de la información.

(1, 2, a), (2, 1, a), (2, 5, b), (3, 3, b), (3, 6, b),
(4, 5, b), (5, 5, c), (6, 3, b), (6, 7, c)

20.9. Suponga que la mitad de las transacciones de una tienda de ropa adquieren vaqueros y un tercio de las transacciones de la tienda adquieren camisetas. Suponga también que la mitad de las transacciones que adquieren vaqueros también adquieren camisetas. Indique todas las reglas de asociación (no triviales) que se puedan deducir de esta información, dando el soporte y la confianza de cada regla.

20.10. Suponga el problema de hallar conjuntos de artículos de gran tamaño.

a. Describa el modo de hallar el soporte de un conjunto dado de conjuntos de elementos mediante una sola exploración de los datos. Suponga que los conjuntos de artículos y la información asociada, como los recuentos, caben en la memoria.

b. Suponga que un conjunto de artículos tiene un soporte menor que j. Demuestre que ningún superconjunto de este conjunto de artículos puede tener un soporte mayor o igual que j.

20.11. Cree un pequeño ejemplo de un conjunto de transacciones que demuestren que aunque muchas transacciones contengan dos artículos (es decir, el conjunto de artículos que contiene los dos artículos tiene un gran soporte), la compra de uno de los artículos puede tener una correlación negativa con la compra del otro.

20.12. La organización de partes, capítulos, secciones y subsecciones de un libro está relacionada con el agrupamiento. Explique la razón y la forma de agrupamiento.

20.13. Proponga la forma de usar las técnicas de minería en un equipo de deporte usando su deporte preferido como ejemplo.

Herramientas

Se dispone de gran variedad de herramientas para cada una de las aplicaciones que se han estudiado en este capítulo. La mayor parte de los fabricantes de bases de datos proporcionan herramientas OLAP como parte de sus sistemas de bases de datos, o como aplicaciones complementarias. Entre ellas están las herramientas OLAP de Microsoft Corp., IBM y Oracle. La herramienta Mondrian es un servidor OLAP de dominio público. Muchas empresas también ofrecen herramientas de análisis para aplicaciones específicas, como la gestión de las relaciones con los clientes.

Los principales fabricantes de bases de datos también ofrecen almacenes de datos acoplados a sus sistemas de bases de datos. Proporcionan ayuda para el modelado de datos, limpieza, carga y consultas. El sitio web www.dwinfocenter.org proporciona información de los productos de almacenes de datos.

También hay una gran variedad de herramientas de minería de datos de propósito general que incluyen las herramientas de minería del SAS Institute, IBM Intelligent Miner y Oracle. Asimismo, existen varias herramientas de minería de datos de fuente abierta, como las muy utilizadas Weka y RapidMiner. El paquete de aplicaciones de inteligencia de negocio de fuente abierta Pentaho dispone de varios componentes como la herramienta ETL, el servidor de OLAP Mondrian y las herramientas de minería de datos basadas en Weka.

Se necesita una gran experiencia para aplicar las herramientas de minería de datos de propósito general para aplicaciones concretas. En consecuencia, se han desarrollado gran número de herramientas de minería para abordar aplicaciones especializadas. El sitio web www.kdnuggets.com proporciona un amplio directorio de software de minería de datos, soluciones, publicaciones, etc.

Notas bibliográficas

Las definiciones de las funciones estadísticas pueden hallarse en los libros de texto habituales de estadística como Bulmer [1979] y Ross [1999].

Los textos de Poe [1995] y Mattison [1996] estudian los almacenes de datos. Zhuge et ál. [1995] describen el mantenimiento de vistas en un entorno de almacenes de datos. Chaudhuri et ál. [2003] describen las técnicas para el encaje difuso en la limpieza de datos, mientras que Sarawagi et ál. [2002] describen un sistema para la desduplicación usando técnicas de aprendizaje activo.

Abadi et ál. [2008] presentan una comparación del almacenamiento orientado a columna y el orientado a fila, incluyendo temas sobre el procesamiento y optimización de consultas.

Witten y Frank [1999] y Han y Kamber [2000] proporcionan cobertura del nivel de los libros de texto de la minería de datos. Mitchell [1997] es autor de un texto clásico sobre aprendizaje automático y trata con detalle las técnicas de clasificación. Fayyad et ál. [1995] presentan un extenso conjunto de artículos sobre el descubrimiento de conocimiento y la minería de datos. Kohavi y Provost [2001] presentan un conjunto de artículos sobre aplicaciones de la minería de datos para el comercio electrónico.

Agrawal et ál. [1993b] proporcionan una primera introducción a la minería de datos en las bases de datos. En Agrawal et ál. [1992] y Shafer et ál. [1996] se describen algoritmos para el cálculo de clasificadores con conjuntos de entrenamiento de gran tamaño; el algoritmo de creación del árbol de decisión descrito en este capítulo se basa en el algoritmo SPRINT de Shafer et ál. [1996]. Cortes y Vapnik [1995] presentaron varios resultados clave sobre las máquinas de vectores de soporte, mientras que Cristianini y Shawe-Taylor [2000] proporcionan cobertura de texto sobre máquinas de vectores de soporte.

Agrawal et ál. [1993a] introducen la noción de reglas de asociación, mientras que Agrawal y Srikant [1994] presentan un algoritmo eficiente para la minería de reglas de asociación. Los algoritmos para la minería de diferentes formas de las reglas de asociación se describen en Srikant y Agrawal [1996a y 1996b]. Chakrabarti et ál. [1998] describen las técnicas para la minería de estructuras temporales sorprendentes. Las técnicas para integrar cubos de datos con la minería de datos se describen en Sarawagi [2000].

Durante mucho tiempo se ha estudiado el agrupamiento en el área de la estadística, y Jain y Dubes [1988] proporcionan cobertura del mismo con nivel de texto. Ng y Han [1994] describen las técnicas del agrupamiento espacial. Las técnicas de agrupamiento para conjuntos de datos de gran tamaño se describen en Zhang et ál. [1996]. Breese et ál. [1998] proporcionan un análisis empírico de diferentes algoritmos para el filtrado cooperativo. Las técnicas para el filtrado cooperativo de los artículos de noticias se describen en Konstan et ál. [1997].

Chakrabarti [2002] y Manning et ál. [2008] proporcionan una descripción con nivel de manual para el estudio de la recuperación de información, incluyendo un extenso tratamiento de las tareas de minería de datos relacionadas con los datos textuales y de hipertexto, tales como la clasificación y el agrupamiento. Chakrabarti [2000] recopila varias técnicas de minería de hipertexto como la clasificación y el agrupamiento de hipertexto.

Recuperación de información

21

Los datos de texto no están estructurados, a diferencia de los datos rígidamente estructurados de las bases de datos relacionales. El término **recuperación de información** suele hacer referencia a la consulta de datos de texto no estructurados. Los sistemas de recuperación de información tienen mucho en común con los sistemas de bases de datos, en especial, el almacenamiento y la recuperación de los datos del almacenamiento secundario. No obstante, el énfasis en el campo de los sistemas de información es diferente del de los sistemas de bases de datos y se centra en problemas como las consultas basadas en palabras clave; la relevancia de los documentos para la consulta y el análisis, la clasificación y el indexado de documentos. Los motores de búsqueda en web actuales llevan más allá el paradigma de la recuperación de documentos y tratan temas más amplios, como la información que mostrar en respuesta a las consultas por palabras clave, para satisfacer las necesidades de información del usuario.

21.1. Descripción general

El campo de la **recuperación de información** se ha desarrollado en paralelo al campo de las bases de datos. En el modelo tradicional usado en el campo de la recuperación de información, esta se organiza en documentos, suponiendo que existe un gran número de documentos. Los datos contenidos en los documentos no están estructurados, no tienen ningún esquema asociado. El proceso de recuperación de información consiste en localizar los documentos importantes, de acuerdo con los datos suministrados por el usuario, como las palabras clave o los documentos de ejemplo.

La web ofrece una manera cómoda de llegar a las fuentes de información y de interactuar con ellas a través de Internet. No obstante, un problema persistente que sufre la web es la explosión de la información almacenada, con pocas indicaciones que ayuden al usuario a localizar la que es interesante. La recuperación de información ha desempeñado un papel esencial para hacer que la web sea una herramienta productiva y útil, especialmente para los investigadores.

Ejemplos tradicionales de sistemas de recuperación de información son los catálogos de bibliotecas en línea y los sistemas de gestión de la documentación en línea como los que almacenan los artículos de los periódicos. Los datos de estos sistemas se organizan como conjuntos de *documentos*; los artículos de los periódicos y las entradas de los catálogos (en los catálogos de las bibliotecas) son ejemplos de documentos. En el contexto de la web, se suele considerar cada página HTML como un documento.

Los usuarios de esos sistemas puede que deseen recuperar un determinado documento o una clase de documentos concreta. Los documentos deseados se suelen describir mediante un conjunto de **palabras clave;** por ejemplo, las palabras clave «sistema de bases de datos» se pueden usar para localizar los libros sobre sistemas de bases de datos y las palabras clave «valores» y «escándalo» se pueden utilizar para localizar artículos sobre escándalos de los mercados de valores. Cada documento tiene asociado un conjunto de palabras clave y se recuperan aquellos cuyas palabras clave contienen las facilitadas por los usuarios.

La recuperación de información basada en las palabras clave no solo se puede usar para recuperar datos de texto, sino también para recuperar otros tipos de datos, como los datos de vídeo o de sonido, que tengan asociadas palabras clave descriptivas. Por ejemplo, una película de vídeo puede tener asociadas palabras clave como el título, el director, los actores, el tipo de película, etc., mientras que una imagen o videoclip puede que tenga etiquetas que son palabras clave que describen la imagen o el videoclip.

Existen varias diferencias entre este modelo y los modelos usados en los sistemas tradicionales de bases de datos.

- Los sistemas de bases de datos gestionan varias operaciones que no se abordan en los sistemas de recuperación de información. Por ejemplo, los sistemas de bases de datos gestionan las actualizaciones y los requisitos transaccionales asociados con el control de concurrencia y de durabilidad. Estos temas se consideran menos importantes en los sistemas de información. De manera parecida, los sistemas de bases de datos gestionan información estructurada organizada mediante modelos de datos relativamente complejos (como el modelo relacional o los modelos de datos orientados a los objetos), mientras que los sistemas de recuperación de información han usado tradicionalmente un modelo mucho más sencillo, en el que la información de la base de datos se organiza simplemente como un conjunto de documentos sin estructurar.
- Los sistemas de recuperación de información afrontan varios problemas que no se han abordado de manera adecuada en los sistemas de bases de datos. Por ejemplo, el campo de la recuperación de información ha tratado los problemas de la gestión de documentos sin estructurar, como la búsqueda aproximada mediante palabras clave y la clasificación de los documentos según el grado estimado de relevancia de cada documento para la consulta.

Además de las palabras clave simples que son solo conjuntos de palabras, los sistemas de recuperación de información suelen permitir las expresiones de consulta formadas mediante palabras clave y las conectivas lógicas *y*, *o* y *no*. Por ejemplo, el usuario puede pedir todos los documentos que contengan las palabras clave «motocicleta *y* mantenimiento», o los documentos que contienen las palabras clave «computadora *o* microprocesador», o incluso los documentos que contienen la palabra clave «computadora *pero no* base de datos». Se da por supuesto que una consulta que contenga palabras clave sin ninguna de las conectivas indicadas usa *y* para conectar las palabras clave de manera implícita.

En la recuperación de **texto completo** se considera que todas las palabras de los documentos son palabras clave. Para los documentos no estructurados, la recuperación de texto completo resulta fundamental, ya que puede que no haya información sobre las pa-

labras del documento que son palabras clave. Se usará la palabra **término** para hacer referencia a las palabras de los documentos, ya que todas las palabras son palabras clave.

En su forma más sencilla, los sistemas de recuperación de información buscan y devuelven todos los documentos que contienen todas las palabras clave de la consulta, si es que no tiene conectivas; las conectivas se manejan de la forma esperada. Los sistemas más sofisticados estiman la relevancia de cada documento para la consulta, de modo que se puedan mostrar ordenados según esta relevancia. Estos sistemas usan información sobre las apariciones de los términos, así como la información de los hipervínculos, para estimar la relevancia.

Los sistemas de recuperación de información, como por ejemplo los motores de búsqueda web, han evolucionado más allá de la recuperación de documentos según un esquema de relevancia. En la actualidad, los motores de búsqueda intentan satisfacer las necesidades de información de los usuarios, juzgando de qué trata un tema, y mostrando no solo las páginas web que se juzgan relevantes, sino también otro tipo de información sobre el tema. Por ejemplo, dado un término de consulta como «cricket», un motor de búsqueda podría mostrar los resultados de los últimos partidos de cricket, en lugar de mostrar los documentos de mayor relevancia relativos al cricket. Otro ejemplo: en respuesta a la consulta «Nueva York», un motor de búsqueda podría mostrar un mapa e imágenes de Nueva York, además de páginas web alusivas a Nueva York.

21.2. Clasificación por relevancia según los términos

El conjunto de documentos que satisface una expresión de consulta puede ser muy grande; en concreto, hay miles de millones de documentos en la web y la mayor parte de las consultas por palabras clave en los motores de búsqueda de la web hallan centenares de miles de documentos que contienen esas palabras clave. La recuperación de texto completo agrava este problema: cada documento puede contener muchos términos, incluso términos que solo se mencionan de pasada se tratan de manera equivalente a documentos en los que el término sí es importante. En consecuencia, puede que se recuperen documentos irrelevantes.

Por tanto, los sistemas de recuperación de información estiman la relevancia de cada documento para la consulta y solo devuelven como respuesta los documentos con una clasificación más elevada. La clasificación por relevancia no es una ciencia exacta, pero hay algunos enfoques ampliamente aceptados.

21.2.1. Clasificación usando TF-IDF

El primer punto que hay que abordar es, dado un término concreto t, averiguar la relevancia para ese término de un documento dado d. Un enfoque es usar el número de apariciones del término en el documento como medida de su relevancia, con la suposición de que es probable que los términos importantes se mencionen muchas veces en el documento. El mero recuento del número de apariciones de un término no suele ser un buen indicador: en primer lugar, el número de apariciones depende de la longitud del documento y, en segundo lugar, puede que un documento que contenga diez apariciones del término no tenga diez veces la relevancia de un documento que contenga una sola aparición.

Un modo de medir $TF(d, t)$, la relevancia del documento d para el término t, es:

$$TF(d, t) = \log\left(1 + \frac{n(d, t)}{n(d)}\right)$$

donde $n(d)$ indica el número de términos del documento y $n(d, t)$ indica el número de apariciones del término t en el documento d. Observe que esta métrica tiene en cuenta la longitud del documento. La relevancia crece con el número de apariciones del término en el documento, aunque no es directamente proporcional al número de apariciones.

Muchos sistemas refinan esta métrica usando otra información. Por ejemplo, si el término aparece en el título, o en la lista de autores o en el resumen, el documento se considera más importante para el término. De manera parecida, si la primera aparición del término se produce muy avanzado el documento, este se puede considerar menos importante que si aparece por primera vez al principio del documento. Los conceptos anteriores pueden formalizarse mediante extensiones de la fórmula que se ha mostrado para $TF(d, t)$. En la comunidad de recuperación de información, la relevancia de un documento para un término se denomina **frecuencia del término** (Term Frequency, TF), independientemente de la fórmula concreta usada.

Una consulta C puede contener varias palabras clave. La relevancia de cada documento para una consulta con dos o más palabras clave se estima combinando las medidas de relevancia del documento para cada palabra clave. Una manera sencilla de combinar las medidas es sumarlas. No obstante, no todos los términos usados como palabras clave son iguales. Suponga que una consulta usa dos términos, uno de los cuales aparece con frecuencia, como «base de datos», y otro que es menos frecuente, como «Silberschatz». Un documento que contenga «Silberschatz» pero no «base de datos» debe clasificarse por encima de otro que contenga «base de datos» pero no «Silberschatz».

Para solucionar este problema se asignan pesos a los términos usando la **frecuencia inversa de los documentos** (Inverse Document Frequency, IDF), definida como:

$$IDF(t) = \frac{1}{n(t)}$$

donde $n(t)$ indica el número de documentos (entre los indexados por el sistema) que contienen el término t. La **relevancia** del documento d para el conjunto de términos C se define como:

$$r(d, Q) = \sum_{t \in Q} TF(d, t) * IDF(t)$$

Esta medida puede refinarse aún más si se permite a los usuarios especificar los pesos $p(t)$ de los términos de la consulta, en cuyo caso los pesos especificados por los usuarios se tienen también en cuenta multiplicando $TF(t)$ por $p(t)$ en la fórmula anterior.

Este enfoque de usar la frecuencia de los términos y la frecuencia inversa de los documentos como medida de la relevancia de estos se denomina enfoque **TF-IDF.**

Casi todos los documentos de texto (en español) contienen palabras como «y», «o», «un», etc. y, por tanto, estas palabras resultan inútiles para propósitos de consulta, ya que su frecuencia inversa de documento es extremadamente baja. Los sistemas de recuperación de información definen un conjunto de palabras, denominadas **palabras de parada,** que contiene aproximadamente cien de las palabras más frecuentes, e ignoran esas palabras al indexar los documentos. Esas palabras no se usan como palabras clave, y se descartan si se hallan entre las que proporciona el usuario.

Otro factor que se tiene en cuenta cuando una consulta contiene varios términos es la **proximidad** de los términos en el documento. Si los términos aparecen cercanos entre sí en el documento, este se clasificará en una posición más elevada que si aparecen muy separados. La fórmula de $r(d, C)$ puede modificarse para tener en cuenta la proximidad.

Dada una consulta C, el trabajo del sistema de recuperación de información es devolver documentos en orden descendente de relevancia para C. Dado que puede haber un número muy grande de documentos que sean relevantes, los sistemas de recuperación de información suelen devolver únicamente los primeros documentos con el grado de relevancia estimada más elevado y permiten que los usuarios soliciten más documentos de manera interactiva.

21.2.2. Recuperación basada en la semejanza

Algunos sistemas de recuperación de información permiten la **recuperación basada en la semejanza.** En este caso, el usuario puede dar al sistema el documento A y pedirle que recupere documentos que sean «semejantes» a A. La semejanza de un documento con otro puede definirse, por ejemplo, con base en los términos comunes. Un enfoque es hallar k términos de A con los valores más elevados de $TF(A, t) * IDF(t)$, y usar esos k términos como consulta para hallar la relevancia de otros documentos. Los términos de la consulta se ponderan con $TF(A, t) * IDF(t)$.

Más genéricamente, la semejanza de los documentos se define mediante la métrica de **semejanza del coseno.** Sean los términos que aparecen en cualquiera de los dos documentos $t_1, t_2, \ldots, t_n$. Sea $r(d, t) = TF(d, t) * IDF(t)$. Entonces, la métrica de semejanza del coseno entre los documentos d y e se define como:

$$\frac{\sum_{i=1}^{n} r(d, t_i) r(e, t_i)}{\sqrt{\sum_{i=1}^{n} r(d, t_i)^2} \sqrt{\sum_{i=1}^{n} r(e, t_i)^2}}$$

Se puede comprobar fácilmente que la métrica de la semejanza del coseno de un documento consigo mismo es 1, mientras que entre dos documentos que no compartan ningún término es 0.

La denominación «semejanza del coseno» procede del hecho de que la fórmula anterior calcula el coseno del ángulo entre dos vectores, cada uno de los cuales representa a uno de los documentos, definido de la manera siguiente. Supóngase que hay n palabras en total en todos los documentos que se vayan a considerar. Se define un espacio n-dimensional, con cada palabra como una de las dimensiones. El documento d se representa mediante un punto de este espacio, con el valor de la coordenada i-ésima del punto igual a $r(d, t_i)$. El vector del documento d conecta el origen (todas las coordenadas iguales a cero) con el punto que representa al documento. El modelo de documentos como puntos y vectores de un espacio n-dimensional se denomina **modelo de espacio vectorial.**

Si el conjunto de documentos semejantes al documento A de la consulta es grande, puede que el sistema solo presente al usuario unos cuantos documentos semejantes, le permita escoger los más destacados e inicie una nueva búsqueda basada en la semejanza con A y con los documentos seleccionados. Es posible que el conjunto de documentos resultante sea lo que el usuario pretendía hallar. Esta idea se denomina **realimentación de la relevancia.**

La realimentación de la relevancia se puede usar también para ayudar a los usuarios a encontrar los documentos importantes de entre un conjunto grande de documentos que coinciden con las palabras clave de consulta dadas. En esta situación se puede permitir que los usuarios identifiquen uno o varios de los documentos devueltos como importantes; el sistema usará los documentos identificados para hallar otros semejantes. Es probable que el conjunto de documentos resultante sea lo que el usuario pretendía hallar. Una alternativa al enfoque de la realimentación de la relevancia es exigir a los usuarios que modifiquen la consulta añadiendo más palabras clave; la realimentación de la relevancia puede resultar más sencilla de usar, además de dar como respuesta un conjunto final de documentos más adecuado.

Para mostrar al usuario un conjunto representativo de documentos cuando el número total es muy grande, el sistema de búsqueda puede agrupar los documentos, basándose en la semejanza del coseno. La agrupación ya se ha descrito en la Sección 20.7, y se han desarrollado varias técnicas para agrupar los conjuntos de documentos. Consulte las notas bibliográficas para más información relativa a la agrupación.

Como caso especial de la similitud, en la web suele haber muchas copias de un mismo documento; podría ocurrir, por ejemplo, si un sitio web copia el contenido de otro. En este caso, no tiene sentido devolver varias copias de un mismo documento con alta clasificación como respuestas diferentes; se deberían detectar los duplicados y devolver como respuesta una única copia.

21.3. Relevancia según los hipervínculos

Los primeros motores de búsqueda web clasificaban los documentos usando solo medidas de relevancia basadas en TF-IDF como las descritas en la Sección 21.2. Sin embargo, estas técnicas presentaban algunas limitaciones cuando se usaban sobre conjuntos muy grandes de documentos, como el conjunto de todas las páginas web. En concreto, muchas páginas web tienen todas las palabras clave especificadas en las consultas típicas del motor de búsqueda; además, algunas de las páginas que los usuarios desean obtener como respuesta, normalmente solo contienen unas cuantas apariciones de los términos de la consulta y no consiguen una puntuación TF-IDF muy elevada.

No obstante, los investigadores pronto se dieron cuenta de que las páginas web contienen información muy importante de la que carecen los documentos de texto sencillo, los hipervínculos. Los hipervínculos se pueden aprovechar para obtener una mejor clasificación por relevancia; en concreto, la clasificación de relevancia de una página está muy influida por los hipervínculos que apuntan *a* esa página. En este apartado se estudia la manera en que se usan los hipervínculos para la clasificación de las páginas web.

21.3.1. Clasificación por popularidad

La idea básica de la **clasificación por popularidad** (también denominada **clasificación por prestigio**) trata de encontrar páginas que sean populares, y clasificarlas por encima de otras páginas que contengan las palabras clave especificadas. Dado que la mayor parte de las búsquedas pretenden hallar información de las páginas más populares, clasificar esas páginas en posiciones más elevadas suele ser una buena idea. Por ejemplo, el término «google» puede aparecer en gran número de páginas, pero la página google.com es el sitio más popular de entre las páginas que contienen el término «google». Por tanto, la página google.com debe clasificarse como la respuesta más importante de las consultas que contienen el término «google».

Las medidas tradicionales de la relevancia de una página, como las basadas en TF-IDF que se vieron en la Sección 21.2, se pueden combinar con la popularidad de la página para obtener una medida global de la relevancia de esta para la consulta. Las páginas con el valor más elevado de relevancia global se devuelven como primeras respuestas de la consulta.

Esto suscita la pregunta de cómo definir y averiguar la popularidad de una página. Una manera es hallar la cantidad de veces que se tiene acceso a la página y usar ese número como medida de la popularidad de ese sitio. Sin embargo, la obtención de esa información es imposible sin la cooperación del sitio, y aunque algunos sitios pueden colaborar a la hora de revelar esta información, es difícil hacerlo para todos los sitios. Además, los sitios pueden mentir sobre su frecuencia de acceso para obtener una clasificación mayor.

Una alternativa muy efectiva es usar los hipervínculos a la página como medida de su popularidad. Muchas personas tienen archivos de marcadores que contienen vínculos a sitios que usan con frecuencia. Se puede deducir que los sitios que aparecen en gran número de archivos de marcadores son muy populares. Los archivos de marcadores se suelen almacenar de manera privada y no suelen ser accesibles en web. No obstante, muchos usuarios tienen páginas web con vínculos a sus páginas web favoritas. Muchos sitios web también tienen vínculos a otros sitios relacionados, que se pueden usar para deducir la popularidad de los sitios vinculados. Los motores de búsqueda de web pueden capturar páginas web (mediante un proceso denominado exploración, *crawling*, que se describe en la Sección 21.7) y analizarlas para encontrar los vínculos entre unas y otras.

Una primera solución para estimar la popularidad de las páginas es usar el número de páginas que enlazan con ellas como medida de su popularidad. Sin embargo, esto en sí mismo tiene el inconveniente de que muchos sitios tienen varias páginas útiles, pero los vínculos externos a menudo solo apuntan a la página raíz de cada sitio. La página raíz, a su vez, tiene vínculos con otras páginas del sitio. Se puede deducir, entonces, erróneamente, que esas otras páginas no son muy populares y tendrán una clasificación baja en las respuestas a las consultas.

Una alternativa es asociar la popularidad con los sitios, en vez de con las páginas. Todas las páginas de un sitio dado tienen la popularidad de ese sitio, y las páginas distintas de la página raíz de los sitios populares se benefician también de la popularidad de esos sitios. Sin embargo, surge la pregunta de qué constituye un sitio. Por lo general, el prefijo de la dirección de Internet de la URL de una página constituye el sitio correspondiente a esa página. No obstante, hay muchos sitios que albergan gran número de páginas muy poco relacionadas entre sí, como los servidores de páginas iniciales de las universidades y de los portales web como groups.yahoo.com o groups.google.com. Para estos sitios, la popularidad de una parte del sitio no implica la popularidad de otras partes del mismo sitio.

Una alternativa más sencilla es permitir la **transferencia de prestigio** desde las páginas populares a las páginas con las que se vinculan. De acuerdo con este esquema, a diferencia del principio «un hombre, un voto» de la democracia, el vínculo desde una página popular x a una página y se considera que confiere más prestigio a la página y que un vínculo desde la página no tan popular z.[1]

Esta definición de popularidad es, de hecho, circular, ya que la popularidad de las páginas queda definida por la popularidad de otras páginas, y puede que haya ciclos de vínculos entre las páginas. No obstante, la popularidad de las páginas puede definirse mediante un sistema de ecuaciones lineales simultáneas, que pueden resolverse a través de técnicas de tratamiento de matrices. Las ecuaciones lineales se definen de manera que tengan una solución única y bien definida.

Es interesante destacar que la idea básica subyacente a las clasificaciones de popularidad es, en realidad, bastante antigua y surgió por primera vez en la teoría de redes sociales desarrollada por los sociólogos de los años cincuenta del siglo XX. En el contexto de las redes sociales, el objetivo era definir el prestigio de las personas. Por ejemplo, el rey de España tiene un elevado prestigio, ya que gran cantidad de gente lo conoce. Si alguien es conocido por varias personas de prestigio, también tendrá un prestigio elevado, aunque no sea conocido por un número de personas tan grande. El uso de conjuntos de ecuaciones lineales para definir las medidas de popularidad también procede de estos trabajos.

1 Esto es parecido, en cierto sentido, a conceder un peso extra al aval que personas famosas (como las estrellas de cine) conceden a ciertos productos, por lo que su importancia es discutible, aunque resulta efectivo y en la práctica se usa mucho.

21.3.2. PageRank

El motor de búsqueda web de Google introdujo **PageRank,** una medida de la popularidad de las páginas basada en la popularidad de las páginas que se vinculan con ellas. El uso de la medida de popularidad PageRank para clasificar las respuestas a las consultas de resultados, que mejoran las de las técnicas de clasificación utilizadas anteriormente, hizo que Google pasara a ser el motor de búsqueda más usado en un periodo de tiempo bastante breve.

PageRank se puede comprender de manera intuitiva si se usa un **modelo de recorrido aleatorio.** Suponga que una persona que navega por la web lleva a cabo un paseo aleatorio (transversal) de las páginas web de la siguiente manera: el primer paso parte de una página web aleatoria y, en cada paso, el navegante aleatorio emprende una de las acciones siguientes. Con una probabilidad δ el navegante salta a una página web escogida aleatoriamente y, con una probabilidad de $1 - \delta$, escoge aleatoriamente uno de los vínculos externos de la página web en la que se halla y sigue ese vínculo. El valor de PageRank de cada página es, por tanto, la probabilidad de que el paseante aleatorio visite la página en cualquier momento dado.

Fíjese en que es más probable que se visiten las páginas a las que apuntan muchas páginas web y, por tanto, tendrán un valor de PageRank más elevado. De manera parecida, las páginas a las que apuntan páginas con un valor de PageRank elevado también tendrán mayor probabilidad de ser visitadas y, por tanto, tendrán un valor de PageRank más elevado.

PageRank se puede definir mediante un conjunto de ecuaciones lineales, de la siguiente manera. En primer lugar, se otorga a cada página web un identificador entero. La matriz de las probabilidades de salto T se define con $T[i, j]$, definida como la probabilidad de que un navegante aleatorio que siga un vínculo externo de la página i siga el vínculo a la página j. Suponiendo que cada vínculo de i tiene igual probabilidad de que lo sigan, $T[i, j] = 1/N_i$, donde N_i es el número de vínculos externos de la página i. La mayor parte de las entradas de T son 0 y se representa mejor mediante una lista de adyacencias. Por tanto, el valor de PageRank $P[j]$ para cada página j se puede definir como:

$$P[j] = \delta / N + (1-\delta) * \sum_{i=1}^{N} \left(T[i, j] * P[i] \right)$$

donde δ es una constante entre 0 y 1 y N es el número de páginas; δ representa la probabilidad de que cada paso del paseo aleatorio sea un salto.

El conjunto de ecuaciones generado de esta manera se suele resolver mediante técnicas iterativas, comenzando con cada $P[i]$ definida como $1/N$. Cada paso de la iteración calcula valores nuevos de cada $P[i]$ usando los valores de P de la iteración anterior. La iteración se detiene cuando la variación máxima entre iteraciones de cualquier valor de $P[i]$ queda por debajo de un valor de corte prefijado.

21.3.3. Otras medidas de la popularidad

Las medidas básicas de la popularidad, como PageRank, desempeñan un papel importante en la clasificación de las respuestas a las consultas, pero no son, de ningún modo, el único factor. La puntuación TF-IDF de cada página se usa para evaluar su relevancia para las palabras clave de la consulta y se debe combinar con la clasificación de popularidad. También se deben tener en cuenta otros factores, para superar las limitaciones de PageRank y de otras medidas de popularidad relacionadas.

Una medida útil de popularidad sería la frecuencia con que se visita un sitio, pero como se ha mencionado anteriormente es difícil de conseguir. Sin embargo, los motores de búsqueda rastrean qué fracción de las veces un usurario hace clic en una página, cuando se devuelve una respuesta. Esta fracción se puede utilizar como medida de la popularidad del sitio. Para medir la fracción de clic, en lugar de proporcionar un enlace directo a la página, el motor de búsqueda proporciona un enlace indirecto dentro del sitio del propio motor de búsqueda, que registra el clic y, de forma transparente, redirige al navegador al enlace original.[2]

Un inconveniente del algoritmo de PageRank es que asigna una medida de popularidad que no tiene en cuenta las palabras clave de la consulta. Por ejemplo, es probable que la página google.com tenga un valor de PageRank muy elevado, ya que muchos sitios contienen vínculos con ella. Suponga que contiene una palabra mencionada de pasada, como «Stanford» (la página de búsqueda avanzada de Google contenía de hecho la palabra Stanford, en algún momento hace años). Una búsqueda con la palabra clave Stanford devolvería google.com como respuesta mejor clasificada, por delante de respuestas más relevantes, como la página web de la Universidad de Stanford.

Una solución a este problema muy utilizada consiste en emplear palabras clave en el texto ancla de los vínculos a una página para evaluar para qué temas es importante la página. El texto ancla de los vínculos consiste en el texto que aparece dentro de la etiqueta a href de HTML.

Por ejemplo, el texto ancla del vínculo:

```
<a href="http://stanford.edu">Stanford University</a>
```

es «Stanford University». Si muchos vínculos a stanford.edu contienen la palabra Stanford en su texto ancla, se puede considerar muy importante para la palabra clave Stanford. El texto cercano al texto ancla también se puede tomar en consideración; por ejemplo, un sitio web puede contener el texto «La página web de la Universidad de Stanford está aquí», pero puede haber usado solo la palabra «aquí» como texto ancla en el vínculo al sitio web de la Universidad de Stanford.

La popularidad basada en el texto ancla se combina con otras medidas de la popularidad y con las medidas TF-IDF para obtener una clasificación global de las respuestas a las consultas, como se describe en la Sección 21.3.5. Como truco de implementación, las palabras en el texto del ancla se tratan a menudo como parte de la página, con una frecuencia de término basada en la popularidad de las páginas en las que aparece el texto del ancla. Entonces, la clasificación TF-IDF tiene en cuenta automáticamente el texto.

Un enfoque alternativo a las palabras clave para definir la popularidad es calcular una medida de la popularidad que *solo* use las páginas que contienen las palabras clave de la consulta, en lugar de calcular la popularidad usando todas las páginas web disponibles. Este enfoque resulta más costoso, ya que el cálculo de las clasificaciones de popularidad tiene que llevarse a cabo de manera dinámica cuando se recibe la consulta, mientras que PageRank se calcula una vez de manera estática y se reutiliza para todas las consultas. Los motores de búsqueda en web que manejan miles de millones de consultas diarias no se pueden permitir emplear tanto tiempo en contestar cada consulta. En consecuencia, aunque este enfoque puede dar mejores respuestas, no se usa mucho.

2 A veces esta dirección queda oculta al usuario. Por ejemplo, cuando se apunta con el ratón en un enlace (como db-book.com) en un resultado de una consulta en Google, el enlace parece señalar directamente al sitio. Sin embargo, al menos hasta el momento de la redacción de este capítulo, cuando se hace clic en un enlace, el código JavaScript asociado a la página realmente reescribe el enlace para ir indirectamente a través del sitio de Google. Si se usa el botón hacia atrás del navegador para volver a la página de consulta, y se vuelve a apuntar al enlace, se hace visible el cambio en la URL explorada.

El algoritmo HITS se basaba en la idea anterior de hallar primero las páginas que contienen las palabras clave de la consulta y calcular luego una medida de la popularidad usando solo ese conjunto de páginas relacionadas. Además, introdujo el concepto de *nodos* y de *autoridades*. Un **nodo** es una página que almacena vínculos con muchas páginas; puede no contener en sí misma información sobre un tema, pero apunta a páginas que sí la contienen. Por el contrario, una **autoridad** es una página que contiene información real sobre un tema, aunque puede que no incluya vínculos con muchas páginas relacionadas. Cada página recibe un valor de prestigio como nodo (*prestigio-nodo*) y otro valor de prestigio como autoridad (*prestigio-autoridad*). Las definiciones de prestigio, como anteriormente, son cíclicas y vienen dadas por un conjunto de ecuaciones lineales simultáneas. Las páginas obtienen mayor prestigio-nodo si apuntan a muchas páginas con un elevado prestigio-autoridad, mientras que reciben un elevado prestigio-autoridad si apuntan a ellas muchas páginas con un elevado prestigio-nodo. Dada una consulta, las páginas con prestigio-autoridad más elevado se clasifican por encima de las demás páginas. Consulte las notas bibliográficas para más información.

21.3.4. Contaminación de los motores de búsqueda

La **contaminación de los motores de búsqueda** hace referencia a la práctica de crear páginas web, o conjuntos de páginas web, diseñadas para obtener una clasificación de relevancia elevada para algunas consultas, aunque los sitios no sean realmente populares. Por ejemplo, puede que un sitio sobre viajes desee obtener una clasificación elevada para las consultas con la palabra clave «viaje». Puede obtener puntuaciones TF-IDF elevadas mediante la repetición de la palabra «viaje» muchas veces en su página.[3] Incluso los sitios sin relación con los viajes, como los sitios pornográficos, pueden hacer lo mismo, y obtener clasificaciones elevadas para las consultas sobre la palabra viaje. De hecho, este tipo de contaminación de TF-IDF era frecuente en los primeros días de la búsqueda en web, y hubo una batalla constante entre este tipo de sitios y los motores de búsqueda que intentaban detectar la contaminación e impedir que consiguieran clasificaciones elevadas.

Los esquemas de clasificación de popularidad como PageRank dificultan la labor de contaminación de los motores de búsqueda, ya que la mera repetición de palabras para obtener puntuaciones TF-IDF elevadas ya no es suficiente. No obstante, incluso estas técnicas se pueden contaminar, mediante la creación de un conjunto de páginas web que se apunten entre sí, lo que incrementa su clasificación de popularidad. Se han propuesto técnicas como el uso de los sitios en lugar de las páginas como unidad de clasificación (con las probabilidades de salto normalizadas correspondientes) para evitar algunas técnicas de contaminación, pero no son del todo eficaces. La guerra entre contaminadores de los motores de búsqueda y los propios motores de búsqueda continúa incluso a día de hoy.

El enfoque de nodos y autoridades del algoritmo HITS es más susceptible de contaminación. El contaminador puede crear una página web que contenga vínculos con autoridades auténticas de un asunto dado y obtener en consecuencia una puntuación de nodo elevada. Además, la web del contaminador incluye vínculos a páginas que desea popularizar, que puede que no tengan ninguna relevancia para ese asunto. Como a estas páginas las apunta una página con una puntuación de nodo elevada, obtienen una puntuación de autoridad elevada pero inmerecida.

3 Las palabras repetidas en las páginas web pueden confundir a los usuarios; los contaminadores pueden abordar este problema entregando páginas diferentes a los motores de búsqueda y al resto de los usuarios, para el mismo URL, o haciendo invisibles las palabras repetidas, por ejemplo, aplicando a esas palabras un formato de fuente blanca pequeña sobre fondo blanco.

21.3.5. Combinación de las medidas TF-IDF con clasificación de popularidad

Se han tratado dos grandes tipos de funciones que sirven para la clasificación, denominados TF-IDF y puntuaciones de popularidad como PageRank. TF-IDF refleja la combinación de varios factores, incluyendo la frecuencia de términos y la frecuencia de documentos inversa, la aparición de un término en un texto ancla de enlace a una página, así como otros varios factores, como la aparición del término en el título, la aparición al principio del documento y el tamaño mayor de la fuente, entre otros.

Cómo combinar estas puntuaciones de una página de cada uno de estos factores para generar una puntuación global es un problema que se puede abordar desde el sistema de recuperación de información. En la primera época de las máquinas de búsqueda, las personas creaban funciones para combinar las puntuaciones en una puntuación global. En la actualidad, las máquinas de búsqueda utilizan técnicas de aprendizaje automático para decidir cómo combinar las puntuaciones. Normalmente, la fórmula de combinación de puntuaciones es fija, pero la fórmula tiene parámetros con los pesos de los distintos factores de puntuación. Usando un conjunto de entrenamiento de resultados de la consulta clasificados por personas, un algoritmo de aprendizaje automático puede encontrar una asignación de pesos para cada factor que genere el mejor rendimiento de la clasificación para varias consultas.

Debe fijarse que la mayoría de los motores de búsqueda no revelan cómo calculan sus clasificaciones de relevancia; creen que revelar sus técnicas de clasificación permitiría a los competidores alcanzarles, y haría el trabajo de los contaminadores más sencillo, haciendo que los resultados fuesen de menor calidad.

21.4. Sinónimos, homónimos y ontologías

Considere el problema de encontrar documentos sobre el mantenimiento de motocicletas, usando la consulta «mantenimiento de motocicletas». Suponga que las palabras clave de los documentos son las palabras del título y el nombre de los autores. No se obtendría el documento titulado *Reparación de motocicletas*, ya que la palabra «mantenimiento» no aparece en el título.

Se puede resolver este problema haciendo uso de los **sinónimos.** Cada palabra puede tener definido un conjunto de sinónimos, y la aparición de una palabra puede ser sustituida por la *disyunción* de todos sus sinónimos (incluida la propia palabra). Así, la consulta «motocicleta *y* reparación» puede sustituirse por «motocicleta *y* (reparación *o* mantenimiento)». Esta consulta sí encontraría el documento deseado.

Las consultas basadas en las palabras clave también se ven afectadas por el problema contrario, el de los **homónimos;** es decir, las palabras con varios significados. Por ejemplo, la palabra «objeto» tiene significados diferentes como nombre y como verbo. La palabra «tabla» puede hacer referencia a una pieza de madera o a una tabla de una base de datos relacional.

De hecho, un riesgo de usar los sinónimos para ampliar las consultas es que estos pueden tener, a su vez, significados diferentes. Por ejemplo, «sustento» es sinónimo de uno de los significados de la palabra «mantenimiento», pero tiene un significado diferente del pretendido por el usuario de la consulta «mantenimiento de motocicletas». Se recuperarán los documentos que utilicen un significado implícito alternativo. El usuario se preguntará el motivo de que el sistema considere que alguno de los documentos recuperados (que, por ejemplo, usan la palabra «sustento») era importante, si no contiene ni las palabras clave especificadas por el usuario ni palabras cuyo significado implícito en el documento sea sinónimo de las palabras clave especificadas. Por tanto, no es buena idea usar sinónimos para ampliar las consultas sin comprobarlos antes con el usuario.

Un enfoque más adecuado del problema anterior es que el sistema comprenda el *concepto* que representa cada palabra del documento y, de manera parecida, entienda el concepto que busca el usuario y devuelva los documentos que aborden los conceptos en los que está interesado el usuario. Los sistemas que soportan las **consultas basadas en conceptos** tienen que analizar cada documento para deshacer la ambigüedad de cada palabra del documento y sustituirla por el concepto que representa; la ruptura de la ambigüedad suele hacerse examinando las palabras circundantes en el documento. Por ejemplo, si el documento contiene palabras como base de datos o consulta, es probable que la palabra tabla sea sustituida por el concepto «tabla: datos», mientras que si el documento contiene palabras como muebles, silla o madera cerca de la palabra tabla, esta sea sustituida por el concepto «tabla: carpintería». La ruptura de la ambigüedad basada en las palabras cercanas suele ser más difícil en las consultas de los usuarios, ya que las consultas contienen muy pocas palabras, por lo que los sistemas de consultas basadas en conceptos suelen ofrecer varios conceptos alternativos al usuario, que escoge uno o varios antes de que la búsqueda se reanude.

Las consultas basadas en conceptos tienen varias ventajas; por ejemplo, una consulta en un idioma puede recuperar documentos en otros idiomas, siempre y cuando estén relacionados con el mismo concepto. Se pueden usar luego mecanismos de traducción automática si el usuario no comprende el idioma en que está escrito el documento. No obstante, la sobrecarga del procesamiento de los documentos para deshacer la ambigüedad de las palabras es muy elevada si se trabaja con miles de millones de documentos. Por tanto, los motores de búsqueda de Internet no suelen soportar las consultas basadas en conceptos. No obstante, se han creado sistemas de consultas basadas en conceptos y se han usado para otros conjuntos documentales de gran tamaño.

Las consultas basadas en conceptos se pueden ampliar aún más aprovechando las jerarquías de conceptos. Por ejemplo, suponga que alguien formula la consulta «animales voladores»; los documentos que contengan información sobre los «mamíferos voladores» son, ciertamente, importantes, ya que los mamíferos son animales. Sin embargo, los dos conceptos no son iguales y la mera coincidencia de conceptos no permite que se devuelva el documento como respuesta. Los sistemas de consultas basadas en conceptos pueden soportar la recuperación de documentos basada en jerarquías de conceptos.

Las **ontologías** son estructuras jerárquicas que reflejan las relaciones entre los conceptos. La más frecuente es la relación **es un/a;** por ejemplo, un leopardo es un mamífero, y un mamífero es un animal. También son posibles otras relaciones, como *parte-de*; por ejemplo, el ala del avión es parte del avión.

El sistema WordNet define gran variedad de conceptos con palabras asociadas (denominadas *synset*, conjunto de sinónimos —synonym set—, en la terminología WordNet). Las palabras asociadas a un synset son sinónimos del concepto; por supuesto, cada palabra puede ser sinónima de varios conceptos diferentes. Además de los sinónimos, WordNet define homónimos y otras relaciones. En concreto, las relaciones *es un/a* y *parte de* conectan conceptos y definen de hecho una ontología. El proyecto Cyc fue otro intento de crear ontologías.

Además de las ontologías que abarcan todo el idioma, se han definido ontologías para áreas concretas con el objetivo de tratar la terminología importante para esas áreas. Por ejemplo, se han creado ontologías para normalizar los términos usados en algunos negocios; se trata de un paso importante en la creación de una infraestructura normalizada para el tratamiento del procesamiento de pedidos y otros flujos de datos internos de las organizaciones. A modo de ejemplo, considere una compañía de seguros médicos que necesita informes de los hospitales que contengan información de

diagnóstico y tratamiento. Una ontología que ayude a normalizar los términos del hospital ayuda al personal de este a entender los informes sin ambigüedades. Esto puede ayudar de forma importante en el análisis de los informes, por ejemplo para realizar un seguimiento de cómo se producen muchos casos de una determinada enfermedad en un intervalo de tiempo concreto.

También es posible crear ontologías que relacionen varios idiomas. Por ejemplo, se han creado WordNets para diferentes idiomas, y los conceptos comunes entre los diferentes idiomas se pueden vincular entre sí. Este tipo de sistemas se puede usar para la traducción de textos. En el contexto de la recuperación de información, las ontologías multilingües se pueden usar para implementar búsquedas basadas en los conceptos en documentos escritos en varios idiomas.

El mayor esfuerzo en el uso de ontologías para las consultas basadas en conceptos es la **web semántica.** La web semántica, liderada por el World Wide Web Consortium, consta de una colección de herramientas, normas y lenguajes que permiten a los datos de la web conectarse de acuerdo con su semántica o significado, en lugar de ser un repositorio centralizado. La web semántica se diseña para permitir el mismo tipo de crecimiento descentralizado y distribuido que ha permitido el éxito del World Wide Web. La clave está en la capacidad de integrar varias ontologías distribuidas. En consecuencia, cualquiera con un acceso a Internet puede añadir términos a la web semántica.

21.5. Creación de índices de documentos

Es importante disponer de una estructura de índices efectiva para el procesamiento eficiente de las consultas en los sistemas de recuperación de información. Se pueden localizar los documentos que contienen las palabras clave especificadas de manera efectiva usando un **índice invertido,** que relaciona cada palabra clave C_i con el conjunto S_i de (los identificadores de) los documentos que contienen C_i. Por ejemplo, si los documentos d_1, d_9 y d_{21} contienen el término «Silberschatz», la lista invertida para la palabra clave Silberschatz sería «d_1, d_9, d_{21}». Para dar soporte a la clasificación por relevancia basada en la proximidad de las palabras clave estos índices pueden proporcionar no solo los identificadores de los documentos, sino también una lista de las ubicaciones dentro del documento en las que aparecen las palabras clave. Por ejemplo, si «Silberschatz» aparece en la posición 21 en d_1, en las posiciones 1 y 19 en d_9 y en las posiciones 4, 29 y 46 en d_{21}, la lista invertida con posiciones sería «d_1/21; d_9/1,19; d_{21}/4, 29, 46».

Estos índices hay que almacenarlos en disco, y cada lista S_i puede abarcar varias páginas de disco. Para minimizar el número de operaciones de E/S para la recuperación de cada lista S_i, el sistema intentará guardar cada lista S_i en un conjunto de páginas consecutivas del disco, de modo que toda la lista se pueda recuperar con una sola búsqueda en el disco. Se pueden usar índices de árbol B^+ para asignar cada palabra clave C_i a su lista invertida asociada S_i.

La operación y encuentra los documentos que contienen todas las palabras clave del conjunto de palabras clave especificado C_1, C_2, …, C_n. La operación y se implementa recuperando primero los conjuntos de identificadores de documentos S_1, S_2, …, S_n de todos los documentos que contienen las respectivas palabras clave. La intersección $S_1 \cap S_2 \cap \ldots, \cap S_n$ de los conjuntos de documentos proporciona los identificadores de documentos del conjunto de documentos deseado. La operación o da el conjunto de todos los documentos que contienen al menos una de las palabras clave C_1, C_2, …, C_n. La operación o se implementa calculando la unión, $S_1 \cup S_2 \cup \ldots, \cup S_n$, de los conjuntos. La operación *no* encuentra los documentos que no contienen la palabra clave especificada C_i. Dado un conjunto de identificadores S, se pueden eliminar los documentos que contienen la palabra clave especificada C_i tomando la diferencia $S - S_i$, donde S_i es el conjunto de identificadores de los documentos que contienen la palabra clave C_i.

Dado un conjunto de palabras clave de una consulta, muchos sistemas de recuperación de información no insisten en que los documentos recuperados contengan todas las palabras clave (a menos que se utilice de manera explícita una operación y). En ese caso, se recuperan todos los documentos que contienen como mínimo una de las palabras clave (como en la operación o), pero se clasifican de acuerdo con la medida de su relevancia.

Para usar la clasificación de frecuencia de los términos la estructura de índices también debe conservar el número de veces que aparece cada término en cada documento. Para reducir este esfuerzo se puede usar una representación comprimida con solo unos pocos bits, que resume aproximadamente la frecuencia de aparición de los términos. El índice también debe almacenar la frecuencia de documentos de cada término (es decir, el número de documentos en que aparece cada término).

Si la clasificación de popularidad es independiente del término índice (como ocurre en el caso de PageRank), la lista S_i se puede ordenar según la clasificación de popularidad (y, en segundo lugar, para los documentos con la misma clasificación de popularidad, por identificador de documento). Luego se puede usar una simple mezcla para calcular las operaciones y y o. En el caso de la operación y, si se ignora la contribución de TF-IDF a la puntuación de relevancia y se exige simplemente que el documento contenga las palabras clave dadas, la mezcla puede detenerse una vez obtenidas C respuestas, si el usuario solo pide las C primeras respuestas. En general, los resultados con clasificación de popularidad más elevada (tras incluir la puntuación TF-IDF) probablemente tendrán una clasificación de popularidad elevada y aparecerán cerca de la cabeza de las listas. Se han desarrollado técnicas para estimar las mejores puntuaciones posibles de los resultados restantes, y estos se pueden utilizar para reconocer que las respuestas que aún no se han visto no pueden formar parte de las K respuestas primeras. El procesamiento de las listas puede terminar anticipadamente.

Sin embargo, la ordenación por la clasificación de popularidad no es totalmente efectiva a la hora de evitar recorrer largas listas invertidas, pues ignora la contribución de las puntuaciones TF-IDF. Una alternativa en estos casos es dividir la lista invertida de cada término en dos partes. La primera parte contiene los documentos que tienen una alta puntuación TF-IDF para dicho término; por ejemplo, los documentos en los que el término aparece en el título del documento o en el texto del anclaje que referencia al documento. La segunda parte contiene todos los documentos. Cada parte de la lista se puede ordenar por popularidad o por el identificador del documento. Dada una consulta, la mezcla de las primeras partes de la lista para cada término probablemente ofrezca varias respuestas con la mayor puntuación global. Si no se encuentran suficientes respuestas de alta puntuación usando las primeras partes de las listas, se usan las segundas partes para encontrar el resto de resultados. Si un documento obtiene una alta puntuación en TF-IDF, probablemente se encuentre en la mezcla de las primeras partes de las listas. Consulte las notas bibliográficas para más información.

21.6. Medida de la efectividad de la recuperación

Cada palabra clave puede estar incluida en gran número de documentos; por tanto, resulta fundamental tener una representación compacta para mantener bajas las necesidades de espacio del índice. Así, los conjuntos de documentos de cada palabra clave se

conservan en forma comprimida. Para ahorrar espacio de almacenamiento a veces se almacena el índice de modo que la recuperación es aproximada; puede que no se recuperen unos pocos documentos de relevancia (lo que se denomina **rechazo falso** o **falso negativo**), o puede que se recuperen unos pocos documentos sin relevancia (lo que se denomina **falso positivo**). Una buena estructura de índice no tiene *ningún* rechazo falso, pero puede que permita algunos falsos positivos; el sistema puede filtrarlos posteriormente mirando las palabras clave que contienen realmente. En el indexado de webs los falsos positivos tampoco son deseables, ya que puede que el documento real no sea accesible rápidamente para su filtrado.

Para medir la calidad con que los sistemas de recuperación de información pueden contestar las consultas se usan dos métricas. La primera, la **precisión,** mide el porcentaje de documentos recuperados que son verdaderamente importantes para la consulta. La segunda, la **recuperación** (*recall*), mide el porcentaje de documentos importantes para la consulta que se ha recuperado. Lo ideal sería que ambas fueran del cien por cien.

La precisión y la recuperación también son medidas importantes para entender la calidad de una determinada estrategia de clasificación de los documentos. Las estrategias de clasificación pueden dar lugar a falsos negativos y a falsos positivos, pero en un sentido sutil.

- Pueden producirse falsos negativos al clasificar los documentos porque los documentos importantes obtengan clasificaciones bajas. Si el sistema obtiene todos los documentos hasta incluir los que tienen clasificaciones muy bajas, habría muy pocos falsos negativos. Sin embargo, los usuarios rara vez miran más allá de las primeras decenas de documentos devueltos y, por tanto, puede que pasen por alto documentos importantes porque no estén clasificados entre los primeros.

  Lo que se considera un falso negativo depende del número de documentos que se examinen. Por tanto, en lugar de tener un solo número como medida de la recuperación, se puede medir la recuperación como función del número de documentos obtenidos.
- Pueden producirse falsos positivos porque documentos sin relevancia obtengan clasificaciones más elevadas que algunos documentos importantes. Esto también depende del número de documentos que se examinen. Una posibilidad es medir la precisión como función del número de documentos extraídos.

Una opción mejor y más intuitiva para la medida de la precisión es medirla como función de la recuperación. Con esta medida combinada tanto la precisión como la recuperación pueden calcularse en función del número de documentos, si hace falta.

Por ejemplo, se puede decir que con una recuperación del cincuenta por ciento la precisión es del setenta y cinco por ciento, mientras que con una recuperación del setenta y cinco por ciento la precisión ha caído hasta el sesenta por ciento.

En general, se puede dibujar una gráfica que relacione la precisión con la recuperación. Estas medidas pueden calcularse para las diferentes consultas y promediarse para un conjunto de consultas en las pruebas de homologación de la calidad de las consultas.

Otro problema más de la medida de la precisión y de la recuperación consiste en el modo de definir los documentos que son realmente importantes y los que no lo son. De hecho, decidir si un documento es importante o no exige entender el lenguaje natural y las intenciones tras la consulta. Los investigadores, por tanto, han creado conjuntos de documentos y de consultas y han marcado manualmente los documentos como importantes o no para las consultas. Con estos conjuntos de documentos se pueden ejecutar diferentes sistemas de clasificación para medir la precisión y la recuperación medias de varias consultas.

21.7. Robots de búsqueda e indexación en web

Los programas denominados **web crawlers** (batidores o robots web) localizan y reúnen información de la web. Siguen de manera recursiva los hipervínculos presentes en los documentos conocidos para hallar otros documentos. Los robots empiezan con un conjunto inicial de URL, que se puede crear manualmente. Se obtienen de la web las páginas indicadas por dichos enlaces URL. A continuación los robots localizan todos los enlaces URL en dichas páginas y los añaden al conjunto de URL a examinar, si no se han obtenido anteriormente o se han añadido al conjunto de las URL a examinar. Este proceso se repite obteniendo todas las páginas del conjunto de búsqueda y procesando los enlaces de dichas páginas de la misma forma. Repitiendo el proceso, eventualmente se obtendrían todas las páginas que se pueden alcanzar mediante una secuencia de enlaces desde el conjunto inicial de URL.

Dado que el número de documentos de web es muy grande, no es posible recorrer toda la web en un periodo corto de tiempo; y, de hecho, todos los motores de búsqueda cubren únicamente algunas partes de la web, no toda ella, y sus robots pueden tardar semanas o meses en llevar a cabo un solo recorrido de todas las páginas que abarcan. Suele haber muchos procesos que se ejecutan en varias máquinas implicadas en los recorridos. Una base de datos almacena el conjunto de vínculos (o de sitios) que hay que recorrer; esa base de datos asigna los vínculos que parten de este conjunto a cada proceso robot. Los nuevos vínculos encontrados durante cada recorrido se añaden a la base de datos, y se pueden recorrer posteriormente si no se recorren de manera inmediata. Las páginas halladas durante cada recorrido también se pasan al sistema de cálculo de prestigio y de indexado, que puede que se ejecute en una máquina diferente. Hay que volver a obtener las páginas (es decir, volver a seguir los vínculos) de manera periódica para obtener información actualizada y descartar los sitios que ya no existen, de modo que la información del índice de búsqueda se mantenga razonablemente actualizada.

Consulte las referencias de la bibliografía para conocer más detalles sobre los recorridos en la web, como las secuencias de enlaces infinitos creados mediante la generación dinámica de páginas (llamada **trampa de araña**), la priorización de búsqueda de páginas y cómo asegurar que los sitios web no se ven desbordados por una ráfaga de peticiones de un robot.

Las páginas que se obtienen durante los recorridos se trasladan a un sistema de cálculo de prestigio e indexación, que puede encontrarse en una máquina diferente. Los propios sistemas de cálculo de prestigio y de indexado se ejecutan en paralelo en varias máquinas. Las páginas se pueden descartar después de calcular el prestigio y añadirlas al índice; sin embargo, normalmente se guarda una copia en el motor de búsqueda, para que los usuarios puedan acceder de forma rápida a una copia de la página, incluso si el sitio web original ya no está accesible.

No es buena idea añadir las páginas al mismo índice que se usa para las consultas, ya que eso exige el control de concurrencia del índice, y afecta al rendimiento de las consultas y de las actualizaciones. En su lugar, se usa una copia del índice para responder a las consultas mientras otra copia se actualiza con las páginas recién recorridas. A intervalos periódicos, se intercambian las copias y se actualiza la más antigua mientras la copia nueva se usa para las consultas.

Para conseguir tasas de consultas muy elevadas se pueden mantener los índices en la memoria principal y usar varias máquinas; el sistema encamina de manera selectiva las consultas a las diferentes máquinas para equilibrar la carga de trabajo entre ellas. Los motores de búsqueda más populares suelen tener decenas de miles de máquinas que llevan a cabo las diferentes tareas de exploración, indexado y respuesta a las consultas de los usuarios.

Los robots web dependen de todas las páginas relevantes que se puedan alcanzar usando hiperenlaces. Sin embargo, muchos sitios contienen grandes colecciones de datos que puede que no estén disponibles desde hiperenlaces. Estos sitios suelen disponer de una interfaz de búsqueda en la que el usuario introduce términos o selecciona opciones de menús para obtener los resultados. Como ejemplo, una base de datos de información aérea normalmente utiliza este tipo de interfaz, sin ningún hipervínculo a las páginas con la información sobre vuelos. En consecuencia, la información de estos sitios no está accesible a los robots web. La información de estos sitios a veces se denomina información de la **web profunda.**

Los **robots de la web profunda** extraen esta información probando con términos que tengan sentido, o eligiendo opciones de los menús en esas interfaces de búsqueda. Introduciendo todos los términos posibles/opciones y ejecutando la interfaz de búsqueda pueden extraer las páginas con los datos que no podrían haber obtenido de otra forma. Las páginas obtenidas por los robots de la web profunda se pueden indexar de la misma forma que las páginas web normales. El motor de búsqueda de Google, por ejemplo, incluye resultados de robots de la web profunda.

21.8. Recuperación de información: más allá de la clasificación de las páginas

Los sistemas de recuperación de información se diseñaron originalmente para encontrar documentos de texto relacionados con las consultas, y se extendieron posteriormente para encontrar páginas en la web. Las personas utilizan los motores de búsqueda para muchas tareas diferentes, desde localizar un sitio web que quieren utilizar a objetivos más amplios, como encontrar información sobre un tema de interés. Los motores de búsqueda web se han convertido en extremadamente buenos en la tarea de localizar los sitios web que quiere visitar un usuario. La tarea de proporcionar información sobre un tema es mucho más costosa y se estudiarán algunos enfoques en esta sección.

También existe una necesidad creciente de sistemas que intenten comprender los documentos (en un grado limitado) y contestar preguntas de acuerdo con esa comprensión (limitada). Un enfoque es la creación de información estructurada a partir de documentos no estructurados y responder a preguntas de acuerdo con esa información estructurada. Otro enfoque aplica las técnicas del lenguaje natural para encontrar documentos relacionados con la pregunta (planteada en lenguaje natural) y devolver fragmentos importantes de los documentos como respuesta a esa pregunta.

21.8.1. Diversidad de resultados de las consultas

En la actualidad, los motores de búsqueda no solo devuelven una lista clasificada de páginas web relevantes a una consulta. También devuelven resultados de imágenes y vídeos. Además, existen multitud de sitios que proporcionan contenido que cambia dinámicamente, como las clasificaciones deportivas o los sitios del mercado de valores. Para obtener información actualizada de esos sitios, los usuarios deberían hacer clic primero en el resultado de la búsqueda. Sin embargo, los motores de búsqueda han creado «gadgets» que recogen datos de un determinado dominio, como los deportes, los precios de las acciones o las condiciones meteorológicas, y les dan formato de forma gráfica para mostrar los resultados de la búsqueda. Los motores de búsqueda tienen que clasificar el conjunto de gadgets disponibles en términos de relevancia para una consulta y mostrar los más relevantes, junto con páginas web, imágenes, vídeos y otro tipo de resultados. Por tanto, el resultado de una consulta tiene diversos conjuntos de tipos de resultados.

Los términos de búsqueda suelen ser ambiguos. Por ejemplo, una consulta como «eclipse» puede referirse a un eclipse solar o de luna, o al entorno de desarrollo integrado (IDE) de nombre Eclipse. Si todas las páginas con mayor clasificación para el término «eclipse» se refieren al IDE, un usuario que busque información sobre los eclipses de sol o de luna puede quedar muy insatisfecho. Los motores de búsqueda, por tanto, intentan proporcionar un conjunto de resultados que sean *diversos* en cuanto al tema, para minimizar la posibilidad de que un usuario quede insatisfecho. Para ello, en el momento de la indexación el motor de búsqueda debe desambiguar el sentido en el que una palabra se usa en una página; por ejemplo, debe decidir si el uso de la palabra «eclipse» en la página se refiere al IDE o al fenómeno astronómico. Entonces, dada una consulta, el motor de búsqueda intenta proporcionar resultados que sean relevantes según el sentido común con que se utilizan las palabras de la búsqueda.

Los resultados que se obtienen de una página web se deben resumir como un **fragmento,** en un resultado de consulta. Tradicionalmente, los motores de búsqueda proporcionan unas pocas palabras de las que rodean las palabras clave de la consulta como un fragmento que ayuda a identificar el contenido de la página. Sin embargo, existen dominios en los que el fragmento se puede generar de forma mucho más significativa. Por ejemplo, si un usuario consulta sobre un restaurante, el motor de búsqueda puede generar un fragmento que contenga la valoración del restaurante, el número de teléfono y un enlace al mapa, además de un vínculo con la página principal del restaurante. Estos fragmentos especializados se suelen generar de resultados a partir de una base de datos, por ejemplo, de restaurantes.

21.8.2. Extracción de información

Los sistemas de **extracción de información** convierten la información de forma textual a otra forma más estructurada. Por ejemplo, un anuncio inmobiliario puede describir los atributos de una casa en forma de texto, como «casa en Queens con dos dormitorios y tres baños, un millón de euros», de la que el sistema de extracción de la información puede extraer atributos como el número de dormitorios, el de baños, el coste y la zona. El anuncio original podría contener términos como «2D», o «dos dorm.» para indicar dos dormitorios. La información así extraída se puede usar para estructurar la información de la forma habitual. Por tanto, un usuario puede especificar que está interesado en casas de dos dormitorios y el sistema debería ser capaz de devolver todas las casas relevantes de dos dormitorios de acuerdo con los datos estructurados, independientemente de los términos utilizados en el anuncio.

Una organización que mantenga una base de datos de información sobre empresas puede usar un sistema de extracción de la información para extraer de manera automática la información de los artículos de los periódicos; la información extraída se relacionará con las modificaciones en los atributos que interesan, como las renuncias, los ceses o los nombramientos de los ejecutivos de las empresas.

Otro ejemplo: los motores de búsqueda diseñados para encontrar artículos de investigación, como Citeseer y Google Scholar, exploran la web para obtener documentos que probablemente sean artículos de investigación. Examinan algunas características de los documentos obtenidos, como la presencia de palabras como «bibliografía», «referencias» y «resumen», para juzgar si un documento es, de hecho, un artículo de investigación. Entonces, extraen el título, la lista de autores y las citas del final del artículo, utilizando técnicas de extracción de información. La información de citas extraída se puede usar para enlazar los artículos con los que cita, o con artículos que le citan a él; estos enlaces de citas entre artículos pueden resultar muy útiles para los investigadores.

Se han creado varios sistemas para la extracción de información para aplicaciones especializadas. Usan técnicas lingüísticas, estructura de páginas y reglas definidas por los usuarios para dominios concretos, como los anuncios inmobiliarios o las publicaciones de investigación. Para dominios limitados, como un sitio web específico, es posible que una persona indique los patrones que se deben utilizar para extraer información. Por ejemplo, en un sitio concreto, un patrón como «precio <número>, donde <número>» indica cualquier número, puede casar con las localizaciones en las que se indica el precio. Estos patrones se pueden crear de forma manual para un número limitado de sitios web.

Sin embargo, con la escala de la web con millones de sitios, la creación manual de estos patrones no es factible. Las técnicas de aprendizaje automático, que pueden aprender estos patrones a partir de ejemplos de entrenamiento, se suelen utilizar para automatizar el proceso de la extracción de información.

La extracción de información normalmente comete errores en una pequeña fracción de la información extraída, habitualmente porque algunas páginas tienen la información en un formato que coincide sintácticamente, pero no se indica realmente un valor (como el precio). La extracción de información usa patrones simples, que de forma separada casan con partes de una página y son muy propensos a errores. Las técnicas de aprendizaje automático pueden realizar un análisis más sofisticado, basado en las interacciones entre patrones, para minimizar los errores en la información extraída, mientras se maximice la cantidad de información extraída. Consulte la bibliografía para más información.

21.8.3. Respuestas a las preguntas

Los sistemas de recuperación de información se centran en la búsqueda de documentos importantes para cada consulta. Sin embargo, la respuesta a una consulta dada puede hallarse solo en parte de un documento, o en partes pequeñas de varios documentos. Los sistemas de **respuestas a las preguntas** intentan ofrecer respuestas directas a las preguntas planteadas por los usuarios. Por ejemplo, una pregunta de la forma «¿Quién mató a Lincoln?» se responde mejor con una línea que diga «John Wilkes Booth disparó a Abraham Lincoln en 1865». Fíjese que la respuesta no contiene realmente las palabras «mató» ni «quién», pero el sistema deduce que «quién» puede responderse con un nombre, y «mató» está relacionado con «disparó».

Los sistemas de respuestas a las preguntas que se centran en la información de la web suelen generar una o más consultas de palabras clave a partir de la pregunta formulada, formulan las consultas de palabras clave a los motores de búsqueda de la web y analizan los documentos devueltos para hallar fragmentos de esos documentos que respondan a la pregunta. Para generar las consultas de palabras clave y para encontrar los fragmentos importantes de los documentos se usan varias técnicas lingüísticas y heurísticas.

Un problema con las respuestas a las preguntas es que distintos documentos pueden dar respuestas diferentes a una pregunta. Por ejemplo, la pregunta «¿Cuánto mide una jirafa?», puede tener respuesta en distintos documentos con diferentes alturas. Estas respuestas forman una distribución de valores y el sistema de respuestas a las preguntas puede elegir el valor promedio, o el valor mediano de la distribución de respuestas; para reflejar el hecho de que el sistema no se espera que sea preciso, el sistema puede devolver el promedio junto la desviación estándar (por ejemplo, el promedio es de 5 metros con una desviación de 70 centímetros), o puede devolver un intervalo según el promedio y la desviación estándar (por ejemplo, entre 4,3 m y 5,7 m).

Los sistemas de respuestas a las preguntas de la generación actual están limitados en potencia, ya que no entienden realmente las preguntas ni los documentos que responden a las preguntas. Sin embargo, son útiles para ciertas preguntas simples.

21.8.4. Consulta de datos estructurados

Los datos estructurados se representan principalmente en forma relacional o en XML. Algunos sistemas permiten las consultas por palabra clave en datos relacionales o en XML (consulte el Capítulo 23). Un asunto muy común entre estos sistemas trata de encontrar los nodos (tuplas o elementos de XML) que contengan las palabras clave especificadas, así como los caminos que los conectan (o los ancestros comunes en el caso de datos XML).

Por ejemplo, una consulta como «Zhag Katz», en la base de datos de la universidad puede encontrar el *nombre* «Zhang» en una tupla de *estudiante* y el *nombre* «Katz» en una tupla *profesor*, así como un camino a través de la relación *tutor* que conecta las dos tuplas. Otros caminos, como que el estudiante «Zhang» está matriculado en una asignatura que imparte «Katz», también se pueden encontrar en respuesta a esta consulta. Estas consultas se pueden utilizar para la navegación y consulta ad hoc cuando el usuario no conoce el esquema exacto y no quiere realizar el esfuerzo de escribir una consulta en SQL que defina esta. De hecho, no parece razonable esperar que los usuarios escriban sus propias consultas en un lenguaje de consultas estructurado, cuando la consulta por palabras clave es bastante natural.

Como las consultas no están totalmente definidas, pueden aparecer distintos tipos de respuestas, que hay que clasificar. Se han propuesto diversas técnicas para clasificar las respuestas de este tipo, de acuerdo a la longitud de los caminos que conectan, y en técnicas de asignación de direcciones y pesos a las aristas. También se han propuesto técnicas de asignación de popularidad a las tuplas y elementos de XML según los enlaces sean con clave externa o enlaces IDREF. Consulte las notas bibliográficas para más información sobre la búsqueda de palabras clave en datos relaciones y en XML.

21.9. Directorios y categorías

El usuario típico de una biblioteca puede usar un catálogo para encontrar el libro que busca; no obstante, cuando coja el libro del estante es probable que *hojee* otros libros que estén próximos. Las bibliotecas organizan los libros de modo que los títulos relacionados se guarden cerca unos de otros. Por tanto, puede que un libro que esté físicamente cerca del libro deseado también sea interesante, lo que hace que también merezca la pena para los usuarios hojear esos libros.

Para guardar cerca unos de otros los libros relacionados, las bibliotecas usan una **jerarquía de clasificación.** Los libros de ciencia se clasifican juntos. Dentro de este conjunto de libros hay una clasificación más detallada, que organiza por un lado los libros de informática, por otro los de matemáticas, etc. Dado que hay una relación entre las matemáticas y la informática, hay conjuntos importantes de libros que se guardan físicamente cerca. En otro nivel diferente de la jerarquía de clasificación, los libros de informática se dividen en subáreas, como los sistemas operativos, los lenguajes y los algoritmos. La Figura 21.1 muestra una jerarquía de clasificación que pueden usar las bibliotecas. Como los libros solo se pueden guardar en un sitio, cada libro de la biblioteca se clasifica exactamente en un punto de la jerarquía de clasificación.

En los sistemas de recuperación de información no hace falta almacenar cerca los documentos relacionados. No obstante, estos sistemas necesitan *organizar lógicamente los documentos* para permitir su exploración. Por tanto, estos sistemas pueden usar una jerarquía de clasificación parecida a la que usan las bibliotecas y, al mostrar un documento concreto, también pueden mostrar una breve descripción de los documentos que se hallan cercanos en la jerarquía.

En los sistemas de recuperación de información no hace falta guardar cada documento en un solo punto de la jerarquía. Los documentos que traten de matemáticas para informáticos pueden clasificarse en matemáticas y en informática. Lo que se guarda en cada punto es un identificador del documento (es decir, un puntero hacia el documento), y resulta fácil recuperar el contenido del documento mediante su identificador.

Como consecuencia de esa flexibilidad, no solo se puede clasificar cada documento en dos posiciones, sino que también puede aparecer un subárea de la jerarquía de clasificación en dos áreas diferentes. La clase de documento «algoritmos de grafos» puede aparecer tanto en matemáticas como en informática. Por tanto, la jerarquía de clasificación es ahora un grafo acíclico dirigido (GAD), como puede verse en la Figura 21.2. Los documentos de algoritmos de grafos pueden aparecer en una sola posición del GAD, pero se puede llegar a ellos por varios caminos.

Un **directorio** no es más que una estructura de clasificación GAD. Cada hoja del directorio almacena vínculos con documentos del tema representado por la hoja. Los nodos internos también pueden contener vínculos, por ejemplo, a documentos que no se pueden clasificar en ninguno de los nodos hijo.

Para encontrar información sobre un tema los usuarios empiezan en la raíz del directorio y siguen un camino por el GAD hasta alcanzar el nodo que representa el tema deseado. Mientras avanzan por el directorio los usuarios no solo pueden encontrar documentos sobre el tema en el que están interesados, sino que también pueden encontrar documentos relacionados y clases relacionadas en la jerarquía de clasificación. Los usuarios pueden conseguir información nueva explorando los documentos (o las subclases) de las clases relacionadas.

La organización de la enorme cantidad de información disponible en web en una estructura de directorio es una tarea enorme.

- El primer problema es la determinación de cuál debe ser exactamente la jerarquía del directorio.
- El segundo problema es, dado un documento, decidir los nodos del directorio que son categorías importantes para el documento.

Para afrontar el primer problema los portales como Yahoo! tienen equipos de «bibliotecarios de Internet» que sugieren la jerarquía de la clasificación y la perfeccionan continuamente.

El segundo problema también puede afrontarse manualmente con bibliotecarios, o bien quienes mantienen el sitio web pueden decidir el lugar de la jerarquía en que deben ubicarse sus sitios. También existen técnicas para decidir automáticamente la ubicación de un documento calculando la similitud con otros documentos ya clasificados.

Wikipedia, la enciclopedia en línea, ataca el problema de la clasificación en la dirección opuesta. Cada página de Wikipedia tiene una lista de **categorías** a la que pertenece. Por ejemplo, en el año 2009, la página de Wikipedia sobre las jirafas tiene varias categorías, entre ellas «Mamíferos de África». De hecho, la propia categoría «Mamíferos de África» pertenece a la categoría «Mamíferos por geografía» que, a su vez, pertenece a la categoría «Mamíferos», que pertenece a la categoría «Vertebrados», y así sucesivamente. La estructura de categorías es útil para explorar otros elementos de la misma categoría, por ejemplo, para descubrir otros mamíferos de África, u otros mamíferos. Consecuentemente, una consulta que busque mamíferos puede usar la información de categorías para inferir que una jirafa es un mamífero. La estructura de categorías de Wikipedia no es un árbol sino casi un GAD, aunque realmente no lo es, pues existen algunos pocos bucles, lo que probablemente es un reflejo de errores de categorización.

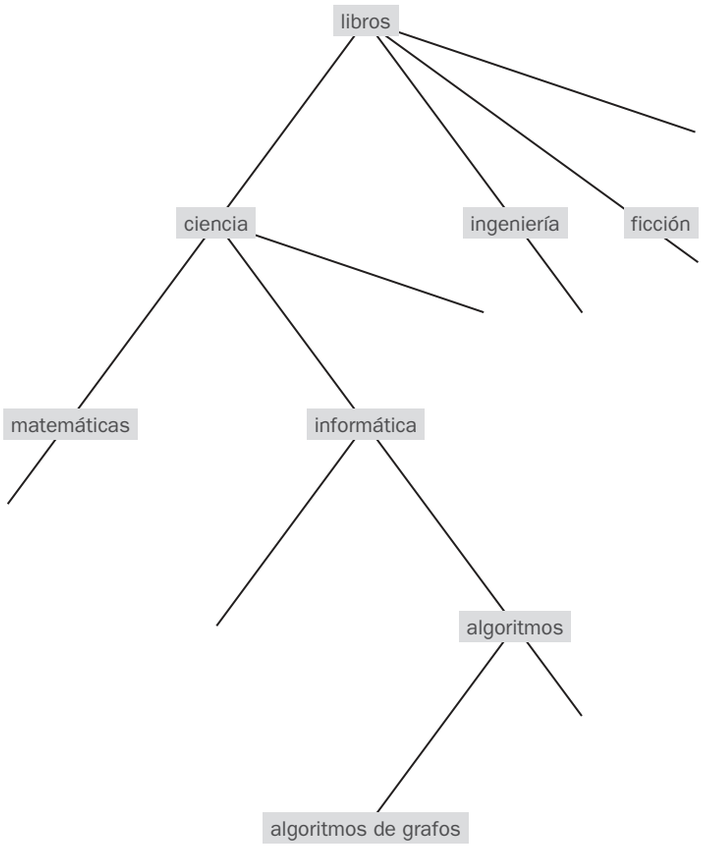


Figura 21.1. Jerarquía de clasificación para el sistema de una biblioteca.

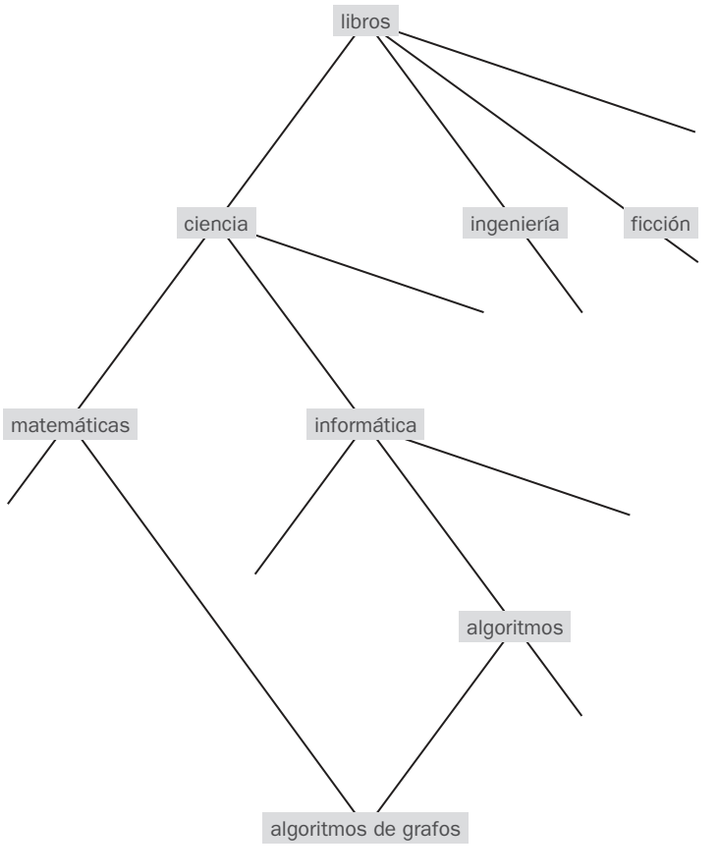


Figura 21.2. GAD de clasificación del sistema de recuperación de información de una biblioteca.

21.10. Resumen

- Los sistemas de recuperación de información se usan para almacenar y consultar datos de texto como son los documentos. Usan un modelo de datos más sencillo que el de los sistemas de bases de datos, pero ofrecen posibilidades de búsqueda más potentes dentro de ese modelo restringido.
- Las consultas intentan localizar los documentos de interés especificando, por ejemplo, conjuntos de palabras clave. La consulta que el usuario tiene en mente no se suele poder formular de manera precisa; por tanto, los sistemas de recuperación de información ordenan las respuestas con base en su posible relevancia.
- La clasificación por relevancia emplea varios tipos de información como:
  - Frecuencia de los términos: la relevancia de cada término para cada documento.
  - Frecuencia inversa de los documentos.
  - Clasificación por popularidad.
- La semejanza de los documentos se usa para recuperar documentos parecidos a un documento de ejemplo. La métrica del coseno se usa para definir la semejanza y se basa en el modelo del espacio vectorial.
- PageRank y la clasificación de nodos y autoridades son dos maneras de asignar prestigio a las páginas con base en los vínculos de cada página. La medida de PageRank se puede comprender de manera intuitiva usando un modelo de recorrido aleatorio. La información del texto ancla también se usa para calcular un concepto de popularidad por palabra clave. Los sistemas de recuperación de la información necesitan combinar las puntuaciones de múltiples factores, como TF-IDF y PageRank, para obtener la puntuación global de una página.
- La contaminación de los motores de búsqueda intenta conseguir una clasificación elevada (no merecida) para una página.
- Los sinónimos y los homónimos complican la tarea de recuperación de la información. Las consultas basadas en conceptos intentan hallar los documentos que contienen los conceptos especificados, independientemente de las palabras exactas (o del idioma) en que se especifiquen. Las ontologías se usan para relacionar los conceptos entre sí, usando relaciones como *es-un* o *parte-de*.
- Los índices invertidos se usan para responder a las consultas de palabras clave.
- La precisión y la recuperación son dos medidas de la efectividad de los sistemas de recuperación de información.
- Los motores de búsqueda recorren la web para encontrar páginas, analizarlas para calcular las medidas de prestigio e indexarlas.
- Se han desarrollado técnicas para extraer información estructurada de los datos de texto, para llevar a cabo consultas basadas en palabras clave sobre datos estructurados y para dar respuestas directas a preguntas sencillas planteadas en lenguaje natural.
- Las estructuras de directorio se usan para clasificar los documentos con otros documentos semejantes.

Términos de repaso

- Sistemas de recuperación de información.
- Búsqueda por palabras clave.
- Recuperación de texto completo.
- Término.
- Clasificación por relevancia.
  - Frecuencia de los términos.
  - Frecuencia inversa de los documentos.
  - Relevancia.
  - Proximidad.
- Recuperación basada en la semejanza.
  - Modelo del espacio vectorial.
  - Métrica de semejanza del coseno.
  - Realimentación de la relevancia.
- Palabras de parada.
- Relevancia usando hipervínculos.
  - Popularidad y prestigio.
  - Transferencia de prestigio.
- PageRank.
  - Modelo de recorrido aleatorio.
- Relevancia basada en el texto ancla.
- Clasificación de nodos y autoridades.
- Contaminación de los índices de búsqueda.
- Sinónimos.
- Homónimos.
- Conceptos.
- Consultas basadas en conceptos.
- Ontologías.
- Web semántica.
- Índice invertido.
- Falso rechazo.
- Falso negativo.
- Falso positivo.
- Precisión.
- Recuperación.
- Robots web.
- Web profunda.
- Diversidad de resultados de consulta.
- Extracción de información.
- Respuestas a preguntas.
- Consulta de datos estructurados.
- Directorios.
- Jerarquía de clasificación.
- Categorías.

Ejercicios prácticos

21.1. Calcule la relevancia (mediante las definiciones correspondientes de la frecuencia de los términos y de la frecuencia inversa de los documentos) de cada uno de los ejercicios prácticos de este capítulo para la consulta «relación SQL».

21.2. Suponga que se desea encontrar los documentos que contengan como mínimo c palabras clave de un conjunto dado de n. Suponga también que se dispone de un índice de palabras clave con una lista (ordenada) de identificadores de los documentos que contienen una palabra clave dada. Indique un algoritmo eficiente para hallar el conjunto de documentos deseado.

21.3. Sugiera la forma de implementar de manera iterativa el cálculo de PageRank dado que la matriz T (incluso en la representación de lista de adyacentes) no cabe en la memoria.

21.4. Sugiera la manera en que se puede indexar un documento que contiene una palabra (como «leopardo») de modo que las consultas que utilicen un concepto más general (como «carnívoro» o «mamífero») lo recuperen de manera eficiente. Se puede suponer que la jerarquía de conceptos no es muy profunda, por lo que cada concepto solo tiene unas cuantas generalizaciones (cada concepto, no obstante, puede tener gran número de especializaciones). También se puede suponer que se dispone de una función que devuelve el concepto de cada palabra del documento. Sugiera también la manera de que las consultas que utilicen conceptos especializados puedan recuperar documentos con conceptos más generales.

21.5. Suponga que se mantienen listas invertidas en bloques, donde cada bloque denota la mejor clasificación de popularidad y las puntuaciones de TF-IDF de los documentos de los demás bloques. Sugiera la manera en que la mezcla de las listas invertidas puede detenerse de manera temprana si el usuario solo desea las primeras C respuestas.

Ejercicios

21.6. Usando una definición sencilla de la frecuencia de cada término, como número de apariciones de cada término en el documento, indique las puntuaciones TF-IDF de cada término del conjunto de documentos que consideramos en este ejercicio y el siguiente.

21.7. Cree un pequeño ejemplo de cuatro documentos de pequeño tamaño, cada uno con su valor de PageRank, y cuatro listas invertidas de los documentos ordenados por PageRank. No hace falta calcular el PageRank, basta con suponer un valor para cada página.

21.8. Suponga que se desea llevar a cabo una consulta mediante palabras clave sobre un conjunto de tuplas de una base de datos, en el que cada tupla tiene unos pocos atributos, cada uno de los cuales solo contiene unas cuantas palabras. ¿El concepto de frecuencia de los términos tiene sentido en este contexto? ¿Y el de frecuencia inversa de los documentos? Justifique sus respuestas. Sugiera también la manera en que se puede definir la semejanza de dos tuplas usando los conceptos de TF-IDF.

21.9. Los sitios web que desean conseguir publicidad pueden unirse a un anillo de web, en el que se crean vínculos con otros sitios del anillo, a cambio de que otros sitios del anillo creen vínculos con ellos. ¿Cuál es el efecto de estos anillos en las técnicas de clasificación por popularidad como PageRank?

21.10. El motor de búsqueda de Google ofrece una característica por la cual los sitios web pueden mostrar anuncios proporcionados por Google. Los anuncios proporcionados se basan en el contenido de cada página. Sugiera la manera en que Google puede escoger los anuncios que debe proporcionar a cada página, dado el contenido de esta.

21.11. Una manera de crear una versión de PageRank específica para palabras clave es modificar el salto aleatorio de modo que solo sean posibles los saltos a páginas que contengan la palabra clave deseada. Por tanto, las páginas que no contengan esa palabra clave pero estén cercanas (en términos de vínculos) a las que sí la contienen obtendrán también una clasificación no nula para esa palabra clave.

a. Escriba las ecuaciones que definen esta versión específica para palabras clave de PageRank.

b. Indique una fórmula para calcular la relevancia de cada página para una consulta que contenga varias palabras clave.

21.12. La idea de clasificación por popularidad mediante hipervínculos se puede ampliar a los datos relacionales y de XML, usando las claves externas y las aristas IDREF en lugar de los hipervínculos. Sugiera la manera de que este esquema de clasificación resulte útil para las siguientes aplicaciones:

a. Una base de datos bibliográfica, que tiene vínculos de los artículos a sus autores y de cada artículo a los artículos a los que hace referencia.

b. Una base de datos de ventas que tiene vínculos de cada registro de ventas a los productos que se han vendido.

Sugiera también el motivo por el que la clasificación por prestigio pueda dar resultados menos significativos en bases de datos de películas que registran qué actores trabajaron en cada película.

21.13. Explique la diferencia entre un falso positivo y un rechazo falso. Si es fundamental que las consultas de recuperación de información no pasen por alto ninguna información importante, explique si es aceptable tener falsos positivos o falsos rechazos. Razone la respuesta.

Herramientas

Google (www.google.com) es actualmente el motor de búsqueda más popular, pero hay otros motores de búsqueda, como Microsoft Bing (www.bing.com) y Yahoo! Search (search.yahoo.com).

El sitio searchenginewatch.com ofrece gran variedad de información sobre los motores de búsqueda. Yahoo! (dir.yahoo.com) y el Proyecto de Directorio Abierto (Open Directory Project, dmoz.org) ofrecen jerarquías de clasificación para los sitios web.

Notas bibliográficas

Manning et ál. [2008], Chakrabarti [2002], Grossman y Frieder [2004], Witten et ál. [1999], y Baeza-Yates y Ribeiro-Neto [1999] ofrecen descripciones propias de libros de texto sobre la recuperación de información. Chakrabarti [2002] y Manning et ál. [2008] ofrecen un tratamiento detallado de los robots web, de las técnicas de clasificación y de las técnicas de minería relacionadas con la recuperación de información, como la clasificación y el agrupamiento de texto.

Brin y Page [1998] describen la anatomía del motor de búsqueda de Google, incluida la técnica PageRank, mientras que una técnica de clasificación basada en nodos y autoridades, denominada HITS, se describe en Kleinberg [1999]. Bharat y Henzinger [1998] presentan una mejora de la técnica de clasificación HITS. Estas técnicas, así como otras técnicas de clasificación basadas en la popularidad (y técnicas para evitar la contaminación de los motores de búsqueda) se describen con detalle en Chakrabarti [2002]. Chakrabarti et ál. [1999] abordan el recorrido enfocado de la web para encontrar páginas relacionadas con un tema concreto. Chakrabarti [1999] ofrece un resumen del descubrimiento de recursos web.

El indexado de los documentos se trata con detalle en Witten et ál. [1999]. Jones y Willet [1997] ofrecen una colección de artículos sobre la recuperación de información. Salton [1989] es uno de los primeros libros de texto sobre los sistemas de recuperación de información. Cierto número de temas prácticos sobre la clasificación e indexación de páginas web, como se realiza en una versión inicial del motor de búsqueda de Google, se tratan en Bin y Page [1998]. Desafortunadamente, no existen detalles públicos sobre cómo se realiza exactamente la clasificación real de ninguno de los motores de búsqueda más importantes.

El sistema Citeseer (citeseer.ist.psu.edu) mantiene una base de datos de gran tamaño de publicaciones (artículos), con vínculos de citas entre ellas, y usa estas citas para clasificar las publicaciones. Incluye una técnica para ajustar la clasificación por citas de acuerdo con la antigüedad de la publicación, para compensar el hecho de que las citas sobre una publicación dada aumentan a medida que pasa el tiempo; sin ese ajuste, los documentos más antiguos tienden a obtener clasificaciones más elevadas de las que merecen realmente. Google Scholar (scholar.google.com) proporciona una base de datos similar de artículos de investigación incorporando citas entre artículos. Es importante darse cuenta de que esos sistemas usan técnicas de extracción de la información para inferir el título y la lista de autores de un artículo, así como las citas del final del artículo. Puede crear vínculos de citas entre artículos según la coincidencia (aproximada) del título y la lista de autores del artículo con el texto de la cita.

La extracción de información y las respuestas a preguntas tienen una historia bastante larga en la comunidad de la inteligencia artificial. Jackson y Moulinier [2002] ofrecen un tratamiento de libro de texto sobre la técnica de procesamiento del lenguaje natural con énfasis en la extracción de la información. Soderland [1999] describe la extracción de la información mediante el sistema WHISK, mientras que Appelt e Israel [1999] ofrecen un tutorial sobre la extracción de información.

La Conferencia anual de Recuperación de Texto (Text Retrieval Conference, TREC) trata sobre varios temas, donde cada uno define un problema y la infraestructura para comprobar la calidad de las soluciones a ese problema. Los detalles sobre TREC se pueden hallar en trec.nist.gov. Puede encontrar más información sobre Wordnet en wordnet.pinceton.edu y globalwordnet.org. El objetivo del sistema Cyc es proporcionar una representación formal de grandes cantidades del conocimiento humano. Su base de conocimiento consta de un gran número de términos, y aserciones sobre cada término. Cyc también incluyen funciones para entender y desambiguar el lenguaje natural. Puede encontrar más información sobre el sistema Cyc en cyc.com y opencyc.org.

La evolución de las búsquedas en la web hacia los conceptos y la semántica en lugar de las palabras clave se trata en Dalvi et ál. [2009]. La Conferencia Internacional sobre Web Semántica (ISWC - International Semantic Web Conference) se celebra anualmente y es una de las mayores conferencias en las que se presentan nuevos desarrollos en web semántica. Puede encontrar más detalles en semanticweb.org.

Agrawal et ál. [2002], Bhalotia et ál. [2002] y Hristidis y Papakonstantinou [2002] tratan la consulta por palabras clave de los datos relacionales. La consulta por palabras clave de los datos de XML se aborda en Florescu et ál. [2000] y Guo et ál. [2003], entre otros.

Parte 7
Bases de datos especiales

Varias áreas de aplicaciones para los sistemas de bases de datos se hallan limitadas por las restricciones del modelo de datos relacional. En consecuencia, los investigadores han desarrollado varios modelos de datos basados en enfoques orientados a objetos para tratar con esos dominios de aplicaciones.

El modelo relacional orientado a objetos, que se describe en el Capítulo 22, combina características del modelo relacional y del modelo orientado a objetos. Este modelo proporciona el rico sistema de tipos de los lenguajes orientados a objetos combinado con las relaciones como base del almacenamiento de los datos. Aplica la herencia a las relaciones, no solo a los tipos. El modelo de datos relacional orientado a objetos permite una migración fácil desde las bases de datos relacionales, lo que resulta atractivo para las diferentes marcas de bases de datos relacionales. En consecuencia, a partir de SQL:1999 la norma incluye varias características orientadas a objetos en su sistema de tipos, aunque que sigue utilizando el modelo relacional como modelo subyacente.

El término base de datos orientada a objetos se utiliza para describir los sistemas de bases de datos que admite el acceso directo a los datos desde lenguajes de programación orientados a objetos, sin necesidad de lenguajes de consultas relacionales como interfaz de las bases de datos. El Capítulo 22 también proporciona una breve visión general de las bases de datos orientadas a objetos.

El lenguaje XML se diseñó inicialmente como un modo de añadir información de marcas a los documentos de texto, pero ha adquirido importancia debido a sus aplicaciones en el intercambio de datos. XML permite representar los datos mediante una estructura anidada y, además, ofrece una gran flexibilidad en su estructuración, lo que resulta importante para ciertos tipos de datos no tradicionales. El Capítulo 23 describe el lenguaje XML y a continuación presenta diferentes formas de expresar las consultas sobre datos representados en XML, incluido el lenguaje de consultas XQuery para XML, que está adquiriendo una gran aceptación.

Bases de datos orientadas a objetos

22

Las aplicaciones tradicionales de bases de datos constan de tareas de procesamiento de datos, como la gestión bancaria y de nóminas, con tipos de datos relativamente sencillos, que se adaptan bien al modelo relacional. A medida que los sistemas de bases de datos se fueron aplicando a un rango más amplio de aplicaciones, como el diseño asistido por computadora y los sistemas de información geográfica, las limitaciones impuestas por el modelo relacional se convirtieron en un obstáculo. La solución fue la introducción de bases de datos basadas en objetos, que permiten trabajar con tipos de datos complejos.

22.1. Descripción general

El primer obstáculo al que se enfrentan los programadores que usan el modelo relacional de datos es el limitado sistema de tipos que admite. Los dominios de aplicación complejos necesitan tipos de datos del mismo nivel de complejidad, como las estructuras de registros anidados, los atributos multivalorados y la herencia, que los lenguajes de programación tradicionales sí admiten. La notación E-R y las notaciones E-R extendidas soportan, de hecho, estas características, pero hay que trasladarlas a tipos de datos de SQL más sencillos. El **modelo de datos relacional orientado a objetos** extiende el modelo de datos relacional ofreciendo un sistema de tipos más rico que incluye tipos de datos complejos y orientación a objetos. Hay que extender de manera acorde los lenguajes de consultas relacionales, en especial SQL, para que puedan trabajar con este sistema de tipos más rico. Estas extensiones intentan conservar los fundamentos relacionales —en especial, el acceso declarativo a los datos— a la vez que extienden la potencia de modelado. Los **sistemas de bases de datos relacionales basadas en objetos;** es decir, los sistemas de bases de datos basados en el modelo objeto-relación, ofrecen un medio de migración cómodo para los usuarios de las bases de datos relacionales que deseen usar características orientadas a objetos.

El segundo obstáculo era la dificultad de acceso a los datos de la base de datos desde los programas escritos en lenguajes de programación como C++ o Java. La mera extensión del sistema de tipos de las bases de datos no resulta suficiente para resolver completamente este problema. Las diferencias entre el sistema de tipos de las bases de datos y el de los lenguajes de programación hace más complicados el almacenamiento y la recuperación de los datos, y se deben minimizar. Tener que expresar el acceso a las bases de datos mediante un lenguaje (SQL), que es diferente del lenguaje de programación, también hace más difícil el trabajo del programador. Es deseable, para muchas aplicaciones, contar con estructuras o extensiones del lenguaje de programación que permitan el acceso directo a los datos de la base de datos, sin tener que pasar por un lenguaje intermedio como SQL.

En este capítulo se explicará primero el motivo del desarrollo de los tipos de datos complejos. Luego se estudiarán los sistemas de bases de datos relacionales orientados a objetos, en concreto con las características introducidas en SQL:1999 y SQL:2003. La mayoría de los productos de bases de datos solo admiten un subconjunto de las características SQL descritas aquí, y para las características admitidas, la sintaxis suele diferir ligeramente respecto a la norma. Es consecuencia de que los sistemas comerciales introdujesen las características relacionales orientadas a objetos en el mercado antes de que se finalizase la norma. Consulte el manual de usuario del sistema de bases de datos que utilice para averiguar las características que soporta.

Posteriormente se estudia el soporte a la persistencia para los datos del sistema de tipos nativo de un lenguaje de programación orientado a objetos. En la práctica se utilizan dos sistemas:

1. Construir un **sistema de bases de datos orientado a objetos;** es decir, un sistema de bases de datos que de forma nativa admita un sistema de tipos orientado a objetos, y permita el acceso directo a los datos desde un lenguaje de programación orientado a objetos usando el sistema de tipos nativo del lenguaje.
2. Convertir automáticamente los datos del sistema de tipos nativo del sistema del lenguaje de programación a una representación relacional y viceversa. La conversión de datos se especifica usando una **asociación objeto-relacional.**

   Se proporciona una breve introducción a ambos enfoques.

Finalmente se describen situaciones en las que el enfoque relacional orientado a objetos es mejor que el enfoque orientado a objetos, y viceversa, y se mencionan criterios para escoger entre los dos.

22.2. Tipos de datos complejos

Las aplicaciones de bases de datos tradicionales tienen conceptualmente tipos de datos simples. Los elementos de datos básicos son registros bastante pequeños y cuyos campos son atómicos, es decir, no contienen estructuras adicionales, y en los que se cumple la primera forma normal (véase el Capítulo 8). Además, solo hay unos pocos tipos de registros.

En los últimos años ha crecido la demanda de formas de abordar tipos de datos más complejos. Considere, por ejemplo, las direcciones. Aunque una dirección completa se puede considerar como un elemento de datos atómico del tipo cadena de caracteres, esa forma de verlo escondería detalles como la calle, la población, la provincia y el código postal, que pueden ser interesantes para las consultas. Por otra parte, si una dirección se representase dividiéndola en sus componentes (calle, población, provincia y código postal) la escritura de las consultas sería más complicada, pues tendrían que mencionar cada campo. Una alternativa mejor es permitir tipos de datos estructurados, que admiten el tipo *dirección* con las subpartes *calle*, *población*, *provincia* y *código_postal*.

Como ejemplo adicional, considere los atributos multivalorados del modelo E-R. Esos atributos resultan naturales, por ejemplo, para la representación de números de teléfono, ya que las personas pueden tener más de un teléfono. La alternativa de la normalización mediante la creación de una nueva relación resulta costosa y artificial para este ejemplo.

Con sistemas de tipos complejos se pueden representar directamente conceptos del modelo E-R, como los atributos compuestos, los atributos multivalorados, la generalización y la especialización, sin necesidad de llevar a cabo una compleja traducción al modelo relacional.

En el Capítulo 8 se definió la *primera forma normal* (1FN), que exige que todos los atributos tengan *dominios atómicos*. Recuerde que un dominio es *atómico* si se considera que los elementos del dominio son unidades indivisibles.

La suposición de la 1FN es natural en los ejemplos de bases de datos que se han considerado. No obstante, no todas las aplicaciones se modelan mejor mediante relaciones en la 1FN. Por ejemplo, en vez de considerar la base de datos como un conjunto de registros, los usuarios de ciertas aplicaciones la ven como un conjunto de objetos (o de entidades). Puede que cada objeto necesite varios registros para su representación. Una interfaz sencilla y fácil de usar necesita una correspondencia de uno a uno entre el concepto intuitivo de objeto del usuario y el concepto de elemento de datos de la base de datos.

Considere, por ejemplo, una aplicación para una biblioteca y suponga que se desea almacenar la siguiente información de cada libro:

- Título del libro.
- Lista de autores.
- Editor.
- Palabras clave.

Es evidente que, si se define una relación para esta información, varios dominios no son atómicos.

- **Autores.** Un libro puede tener una lista de autores, que se puede representar como un array. Pese a todo, puede que se quieran encontrar todos los libros de los que Jones es autor. Por tanto, lo que interesa es una subparte del elemento del dominio «autores».
- **Palabras clave.** Si se almacena un conjunto de palabras clave para cada libro, se espera poder recuperar todos los libros cuyas palabras clave incluyan uno o más términos dados. Por tanto, se considera el dominio del conjunto de palabras clave como no atómico.
- **Editor.** A diferencia de *palabras clave* y *autores*, *editor* no tiene un dominio que se evalúe en forma de conjunto. Sin embargo, se puede considerar que *editor* consta de los subcampos *nombre* y *sucursal*. Este punto de vista hace que el dominio de *editor* no sea atómico.

La Figura 22.1 muestra una relación de ejemplo, *libros*.

título	*array_autores*	*editor*	*palabras_clave*
		(*nombre*, *sucursal*)	
Compiladores	[Smith, Jones]	(McGraw-Hill, NewYork)	{análisis sintáctico, análisis}
Redes	[Jones, Frick]	(Oxford, London)	{Internet, web}

Figura 22.1. Relación *libros* que no está en la 1FN.

título	*autor*	*posición*
Compiladores	Smith	1
Compiladores	Jones	2
Redes	Jones	1
Redes	Frick	2

autores

título	*palabras_clave*
Compiladores	análisis sintáctico
Compiladores	análisis
Redes	Internet
Redes	Web

palabras_clave

título	*nombre_editor*	*sucursal_editor*
Compiladores	McGraw-Hill	New York
Redes	Oxford	London

libros4

Figura 22.2. Versión en la 4FN de la relación *libros*.

Por simplificar se da por supuesto que el título del libro lo identifica de manera unívoca.[1] Se puede representar, entonces, la misma información usando el esquema siguiente, en el que se ha subrayado el atributo clave primaria:

- *autores*(*título*, *autor*, *posición*)
- *palabras_clave*(*título*, *palabra_clave*)
- *libros4*(*título*, *nombre_editor*, *sucursal_editor*)

El esquema anterior satisface la 4FN. La Figura 22.2 muestra la representación normalizada de los datos de la Figura 22.1.

Aunque la base de datos de libros de ejemplo puede expresarse de manera adecuada sin necesidad de usar relaciones anidadas, su uso lleva a un modelo más fácil de comprender. El usuario típico o el programador de sistemas de recuperación de la información piensan en la base de datos en términos de libros que tienen conjuntos de autores, como los modelos de diseño que no se hallan en la 1FN. El diseño de la 4FN exige consultas que reúnan varias relaciones, mientras que los diseños que no se hallan en la 1FN hacen más fáciles muchos tipos de consultas.

Por otro lado, en otras situaciones puede resultar más conveniente usar una representación en la primera forma normal. Por ejemplo, considere la relación *matricula* del ejemplo de la universidad. La relación es de varios a varios entre *estudiantes* y *secciones*. Teóricamente, se podría almacenar un conjunto de estudiantes con cada sección, o un conjunto de secciones con cada estudiante, o ambas cosas. Si se almacenaran ambas cosas, se tendría redundancia de datos (la relación de un estudiante concreto con una sección dada se almacenaría dos veces).

La capacidad de usar tipos de datos complejos, como los conjuntos y los arrays, puede resultar útil en muchas aplicaciones, pero se deberían usar con cuidado.

1 Esta suposición no se cumple en el mundo real. Los libros suelen identificarse por un número ISBN de 10 dígitos que es un identificador único para cada libro publicado.

22.3. Tipos estructurados y herencia en SQL

Antes de SQL:1999 el sistema de tipos de SQL consistía en un conjunto bastante sencillo de tipos predefinidos. SQL:1999 añadió un sistema de tipos extenso a SQL, lo que permite los tipos estructurados y la herencia de tipos.

22.3.1. Tipos estructurados

Los tipos estructurados permiten representar directamente los atributos compuestos en los diagramas E-R. Por ejemplo, se puede definir el siguiente tipo estructurado para representar el atributo compuesto *nombre* con los atributos componentes *nombre* y *apellidos*:

```
create type Nombre as
        (nombre varchar(20),
        apellidos varchar(20))
        final;
```

De manera parecida, el tipo estructurado siguiente puede usarse para representar el atributo compuesto *dirección*:

```
create type Dirección as
        (calle varchar(20),
        ciudad varchar(20),
        códigopostal varchar(9))
        not final;
```

En SQL[2] estos tipos se denominan tipos **definidos por el usuario.** La definición anterior corresponde al diagrama E-R de la Figura 7.4. Las especificaciones **final** y **not final** están relacionadas con la creación de subtipos, que se describirá más adelante, en la Sección 22.3.2.[3]

Ahora se pueden usar esos tipos para crear atributos compuestos en las relaciones con solo declarar que un atributo es de uno de estos tipos. Por ejemplo, se puede crear una tabla *persona* de la siguiente manera:

```
create table persona (
        nombre Nombre,
        dirección Dirección,
        fechaDeNacimiento date);
```

Se puede tener acceso a los componentes de los atributos compuestos usando la notación «punto»; por ejemplo, *nombre.apellidos* devuelve el componente apellidos del atributo nombre. El acceso al atributo *nombre* devolvería un valor del tipo estructurado *Nombre*.

También se puede crear una tabla cuyas filas sean de un tipo definido por el usuario. Por ejemplo, se puede definir el tipo *TipoPersona* y crear la tabla *persona* de la siguiente manera:[4]

```
create type TipoPersona as (
        nombre Nombre,
        dirección Dirección,
        fechaDeNacimiento date)
        not final
create table persona of TipoPersona;
```

Una manera alternativa de definir los atributos compuestos en SQL es usar **tipos de fila** sin nombre. Por ejemplo, la relación que representa la información de una persona se podría haber creado usando tipos de fila de la siguiente manera:

```
create table persona_r (
        nombre row (nombre varchar(20),
                    apellidos varchar(20)),
        dirección row (calle varchar(20),
                       ciudad varchar(20),
                       códigopostal varchar(9)),
        fechaDeNacimiento date);
```

Esta definición es equivalente a la anterior definición de la tabla, salvo en que los atributos *nombre* y *dirección* tienen tipos sin nombre y las filas de la tabla también tienen un tipo sin nombre.

La siguiente consulta muestra la manera de tener acceso a los atributos componentes de los atributos compuestos. La consulta busca el apellido y la ciudad de todas las personas.

```
select nombre. apellidos, dirección. ciudad
from persona;
```

Sobre los tipos estructurados se pueden definir **métodos.** Los métodos se declaran como parte de la definición de los tipos estructurados:

```
create type TipoPersona as (
        nombre Nombre,
        dirección Dirección,
        fechaDeNacimiento date)
        not final
method edadAFecha (aFecha date)
        returns interval year;
```

El cuerpo del método se crea por separado:

```
create instance method edadAFecha (aFecha date)
        returns interval year
        for TipoPersona
begin
        return aFecha − self.fechaDeNacimiento;
end
```

Tenga en cuenta que la cláusula **for** indica el tipo al que se aplica el método, mientras que la palabra clave **instance** indica que el método se ejecuta sobre un ejemplar del tipo *Persona*. La variable **self** hace referencia al ejemplar de *Persona* sobre el que se invoca el método. El cuerpo del método puede contener instrucciones procedimentales, que ya se han visto en la Sección 5.2. Los métodos pueden actualizar los atributos del ejemplar sobre el que se ejecutan.

Los métodos se pueden invocar sobre los ejemplares de los tipos. Si se hubiera creado la tabla *persona* del tipo *TipoPersona*, se podría invocar el método *edadAFecha* ( ) como se muestra a continuación, para calcular la edad de todas las personas.

```
select nombre. apellidos, edadAFecha (current_date)
from persona;
```

En SQL:1999 se usan **funciones constructoras** para crear valores de los tipos estructurados. Las funciones con el mismo nombre que un tipo estructurado son funciones constructoras de ese tipo estructurado. Por ejemplo, se puede declarar una función constructora para el tipo *Nombre* de esta manera:

```
create function Nombre (nombre varchar(20),
                        apellidos varchar(20))
returns Nombre
begin
        set self.nombre = nombre;
        set self.apellidos = apellidos;
end
```

2 Para mostrar nuestra afirmación anterior sobre las implementaciones comerciales que definen su propia sintaxis antes de que la norma se desarrollara, indicar que Oracle requiere que se use la palabra **object** después de **as.**

3 La especificación **final** de *Nombre* indica que no se pueden crear subtipos de *nombre*, mientras que la especificación **not final** de *Dirección* indica que se pueden crear subtipos de *dirección*.

4 La mayoría de los sistemas actuales no distinguen entre mayúsculas y minúsculas, por lo que no permitirían que se usase *nombre* tanto para el nombre de un atributo como para un tipo de datos.

Se puede usar **new** *Nombre* (´John´, ´Smith´) para crear un valor del tipo *Nombre*. Se puede crear un valor de fila haciendo una relación de sus atributos entre paréntesis. Por ejemplo, si se declara el atributo *nombre* como tipo de fila con los componentes *nombre* y *apellidos*, se puede crear para él este valor: (´Ted´, ´Codd´) sin necesidad de función constructora.

De manera predeterminada, cada tipo estructurado tiene una función constructora sin argumentos, que configura los atributos con sus valores predeterminados. Cualquier otra función constructora hay que crearla de manera explícita. Puede haber más de una función constructora para el mismo tipo estructurado; aunque tengan el mismo nombre, deben poder distinguirse por el número y tipo de sus argumentos.

La siguiente instrucción muestra la manera de crear una nueva tupla de la relación *Persona*. Se da por supuesto que se ha definido una función constructora para *Dirección*, igual que la función constructora que se definió para *Nombre*.

insert into *Persona*
values
(**new** *Nombre* (´John´, ´Smith´),
new *Dirección* (´20 Main St´, ´New York´, ´11001´),
date ´22-8-1960´);

22.3.2. Herencia de tipos

Suponga que se tiene la siguiente definición de tipo para las personas:

create type *Persona*
(*nombre* **varchar**(20),
dirección **varchar**(20));

Puede que se desee almacenar en la base de datos información adicional sobre las personas que son estudiantes y sobre las que son profesores. Dado que los estudiantes y los profesores también son personas, se puede usar la herencia para definir en SQL los tipos estudiante y profesor:

create type *Estudiante*
under *Persona*
(*grado* **varchar**(20),
departamento **varchar**(20));

create type *Profesor*
under *Persona*
(*sueldo* **integer**,
departamento **varchar**(20));

Tanto *Estudiante* como *Profesor* heredan los atributos de *Persona*; es decir, *nombre* y *dirección*. Se dice que *Estudiante* y *Profesor* son subtipos de *Persona* y *Persona* es un supertipo de *Estudiante* y de *Profesor*.

Los métodos de los tipos estructurados se heredan por sus subtipos, igual que los atributos. Sin embargo, cada subtipo puede redefinir el comportamiento de los métodos volviendo a declararlos, usando **overriding method** en lugar de **method** en la declaración del método.

La norma de SQL requiere que haya un campo adicional al final de la definición de los tipos, cuyo valor es **final** o **not final.** La palabra clave **final** indica que no se pueden crear subtipos a partir del tipo dado, mientras que **not final** indica que se pueden crear subtipos.

Suponga que se desea almacenar información sobre los profesores ayudantes, que son a la vez estudiantes y profesores, quizás incluso en departamentos diferentes. Esto se puede hacer si el sistema de tipos soporta **herencia múltiple,** por la que se pueden declarar los tipos como subtipos de varios tipos. Tenga en cuenta que la norma de SQL (hasta las versiones SQL:1999 y SQL:2003, al menos) no admite herencia múltiple, aunque puede que sí se incorpore en versiones futuras, por lo que lo trataremos a continuación.

Por ejemplo, si el sistema de tipos dispone de herencia múltiple, se puede definir el tipo para los profesores ayudantes de la manera siguiente:

create type *ProfesorAyudante*
under *Estudiante*, *Profesor*;

ProfesorAyudante heredará todos los atributos de *Estudiante* y de *Profesor*. No obstante, existe un problema, ya que los atributos *nombre*, *dirección* y *departamento* existen tanto en *Estudiante* como en *Profesor*.

Los atributos *nombre* y *dirección* se heredan realmente de una fuente común, *Persona*. Por tanto, no hay ningún conflicto por heredarlos de *Estudiante* y de *Profesor*. Sin embargo, el atributo *departamento* se define por separado en *Estudiante* y en *Profesor*. De hecho, cada profesor ayudante puede ser estudiante en un departamento y profesor en otro. Para evitar conflictos entre las dos apariciones de *departamento*, se pueden rebautizar usando una cláusula **as,** como en la definición del tipo *ProfesorAyudante*:

create type *ProfesorAyudante*
under *Estudiante* **with** (*departamento* **as** *departamento_estudiante)*,
Profesor **with** (*departamento* **as** *departamento_profesor*);

En SQL, como en la mayor parte del resto de los lenguajes, el valor de un tipo estructurado debe tener exactamente un tipo *más concreto*. Es decir, cada valor debe asociarse, al crearlo, con un tipo concreto, denominado su **tipo más concreto.** Mediante la herencia también se asocia con cada uno de los supertipos de su tipo más concreto. Por ejemplo, suponga que una entidad es tanto del tipo *Persona* como del tipo *Estudiante*. Entonces, el tipo más concreto de la entidad es *Estudiante*, ya que *Estudiante* es un subtipo de *Persona*. Sin embargo, una entidad no puede ser de los tipos *Estudiante* y *Profesor*, a menos que tenga un tipo, como *ProfesorAyudante*, que sea subtipo de *Profesor* y de *Estudiante* (lo cual no es posible en SQL, ya que SQL no admite herencia múltiple).

22.4. Herencia de tablas

Las subtablas de SQL se corresponden con el concepto de especialización/generalización de E-R. Por ejemplo, suponga que se define la tabla *personas* de la manera siguiente:

create table *personas* **of** *Persona*;

A continuación se pueden definir las tablas *estudiantes* y *profesores* como **subtablas** de *personas*, de la siguiente manera:

create table *estudiantes* **of** *Estudiante*
under *personas*;

create table *profesores* **of** *Profesor*
under *personas*;

Los tipos de las subtablas (*Estudiante* y *Profesor* en el ejemplo anterior) son subtipos del tipo de la tabla padre (*Persona* en el ejemplo anterior). Por tanto, todos los atributos presentes en *personas* también están presentes en las subtablas *estudiantes* y *profesores*.

Además, cuando se declaran *estudiantes* y *profesores* como subtablas de *personas*, todas las tuplas presentes en *estudiantes* o en *profesores* pasan a estar también presentes de manera implícita en *personas*. Por tanto, si una consulta usa la tabla *personas*, no solo encuentra tuplas directamente insertadas en esa tabla, sino

también tuplas insertadas en sus subtablas; es decir, *estudiantes* y *profesores*. No obstante, esa consulta solo puede tener acceso a los atributos que están presentes en *personas*.

SQL permite encontrar las tuplas que se encuentran en *personas* pero no en sus subtablas usando en las consultas «**only** *personas*» en lugar de *personas*. La palabra clave **only** también se puede usar en las sentencias delete y update. Sin la palabra clave **only,** la instrucción delete aplicada a una supertabla, como *personas*, también borra las tuplas que se insertaron originalmente en las subtablas (como *estudiantes*); por ejemplo, la sentencia:

delete from *personas* **where** *P*;

borrará todas las tuplas de la tabla *personas*, así como de sus subtablas *estudiantes* y *profesores*, que satisfagan *P*. Si se añade la palabra clave **only** a la sentencia anterior, las tuplas que se insertaron en las subtablas no se ven afectadas, aunque satisfagan las condiciones de la cláusula **where.** Las consultas posteriores a la supertabla seguirán encontrando esas tuplas.

Teóricamente, la herencia múltiple es posible con las tablas, igual que con los tipos. Por ejemplo, se puede crear una tabla del tipo *ProfesorAyudante*:

create table *profesores_ayudantes*
of *ProfesorAyudante*
under *estudiantes*, *profesores*;

Como consecuencia de la declaración, todas las tuplas presentes en la tabla *profesores_ayudantes* también se hallan presentes de manera implícita en las tablas *profesores* y *estudiantes* y, a su vez, en la tabla *personas*. Hay que tener en cuenta, no obstante, que SQL no soporta la herencia múltiple de tablas.

Existen varios requisitos de consistencia para las subtablas. Antes de definir las restricciones es necesaria una definición: se dice que las tuplas de una subtabla se **corresponden** con las tuplas de la tabla madre si tienen el mismo valor para todos los atributos heredados. Por tanto, las tuplas correspondientes representan a la misma entidad.

Los requisitos de consistencia de las subtablas son:

1. Cada tupla de la supertabla se puede corresponder, como máximo, con una tupla de cada una de sus subtablas inmediatas.
2. SQL posee una restricción adicional que hace que todas las tuplas que se corresponden entre sí deben proceder de una tupla (insertada en una tabla).

Por ejemplo, sin la primera condición, se podrían tener dos tuplas de *estudiantes* (o de *profesores*) que correspondieran a la misma persona.

La segunda condición excluye que haya una tupla de *personas* correspondiente a una tupla de *estudiantes* y a una tupla de *profesores*, a menos que todas esas tuplas se hallen presentes de manera implícita porque se haya insertado una tupla en la tabla *profesores_ayudantes*, que es subtabla tanto de *profesores* como de *estudiantes*.

Dado que SQL no dispone de herencia múltiple, la segunda condición impide realmente que ninguna persona sea a la vez profesor y estudiante. Aunque se tuviese herencia múltiple, surgiría el mismo problema si estuviera ausente la subtabla *profesores_ayudantes*. Evidentemente, resultaría útil modelar una situación en la que alguien pudiera ser a la vez profesor y estudiante, aunque no se hallara presente la subtabla *profesores_ayudantes*. Por tanto, puede resultar útil eliminar la segunda restricción de consistencia. Hacerlo permitiría que cada objeto tuviera varios tipos, sin necesidad de que tuviera un tipo más concreto.

Por ejemplo, suponga que se vuelve a tener el tipo *Persona*, con los subtipos *Estudiante* y *Profesor*, y la tabla correspondiente *personas*, con las subtablas *profesores* y *estudiantes*. Se puede tener una tupla en *profesores* y una tupla en *estudiantes* correspondientes a la misma tupla en *personas*. No hace falta tener el tipo *ProfesorAyudante*, que es subtipo de *Estudiante* y de *Profesor*. No es necesario crear el tipo *ProfesorAyudante* a menos que se desee almacenar atributos adicionales o redefinir los métodos de manera específica para las personas que sean a la vez estudiantes y profesores.

No obstante, hay que tener en cuenta que, por desgracia, SQL prohíbe las situaciones de este tipo debido al segundo requisito de consistencia. Como SQL tampoco soporta la herencia múltiple, no se puede usar la herencia para modelar una situación en la que alguien pueda ser a la vez estudiante y profesor. En consecuencia, las subtablas de SQL no se pueden usar para representar las especializaciones solapadas de los modelos E-R.

Por supuesto, se pueden crear tablas diferentes para representar las especializaciones o generalizaciones solapadas sin usar la herencia. El proceso ya se ha descrito anteriormente, en la Sección 7.8.6.1. En el ejemplo anterior, se crearían las tablas *personas*, *estudiantes* y *profesores*, en las que las tablas *estudiantes* y *profesores* contendrían el atributo de clave primaria de *Persona*, así como otros atributos específicos de *Estudiante* y de *Profesor*, respectivamente. La tabla *personas* contendría la información sobre todas las personas, incluidos los estudiantes y los profesores. Luego habría que añadir las restricciones de integridad referencial correspondientes para garantizar que los estudiantes y los profesores también se encuentran representados en la tabla *personas*.

En otras palabras, se puede crear una implementación mejorada del mecanismo de las subtablas mediante las características de SQL ya existentes, con algún esfuerzo adicional para la definición de la tabla, así como en el momento de las consultas para especificar las reuniones para el acceso a los atributos necesarios.

Hay que tener en cuenta que SQL define un privilegio denominado **under,** necesario para crear subtipos o subtablas bajo otro tipo o tabla. La razón de ser de este privilegio es parecida a la del privilegio **references.**

22.5. Tipos array y multiconjunto en SQL

SQL soporta dos tipos de conjuntos: arrays y multiconjuntos; los tipos array se añadieron en SQL:1999, mientras que los tipos multiconjunto se agregaron en SQL:2003. Recuerde que un *multiconjunto* es un conjunto no ordenado en el que cada elemento puede aparecer varias veces. Los multiconjuntos son como los conjuntos, salvo que los conjuntos permiten que cada elemento aparezca, como mucho, una vez.

Suponga que se desea registrar información sobre libros, incluido un conjunto de palabras clave para cada libro. Suponga también que se desea almacenar el nombre de los autores de un libro en forma de array; a diferencia de los elementos de los multiconjuntos, los elementos de los arrays están ordenados, de modo que se puede distinguir el primer autor del segundo, etc. El siguiente ejemplo muestra la manera en que se pueden definir en SQL estos atributos valorados como arrays y como multiconjuntos.

create type *Editor* **as**
(*nombre* **varchar**(20);
sucursal **varchar**(20));

create type *Libro* **as**
(*título* **varchar**(20),
array_autores **varchar**(20) **array** [10],
fecha_publicación **date**,
editor Editor,
palabras_clave **varchar**(20) **multiset**);

create table *libros* **of** *Libro*;

La primera sentencia define el tipo denominado *Editor*, que tiene dos componentes: nombre y sucursal. La segunda sentencia define el tipo estructurado *Libro*, que contiene un *título*, un *array_autores*, que es un array de hasta diez nombres de autor, una fecha de publicación, un editor (del tipo *Editor*) y un multiconjunto de palabras clave. Finalmente, se crea la tabla *libros*, que contiene las tuplas del tipo *Libro*.

Tenga en cuenta que se ha usado un array, en lugar de un multiconjunto, para almacenar el nombre de los autores, ya que el orden de los autores suele tener cierta importancia, mientras que se considera que el orden de las palabras asociadas con libro no es significativo.

En general, los atributos multivalorados de un esquema E-R se pueden asignar en SQL a atributos valorados como multiconjuntos; si el orden es importante, se pueden usar los arrays de SQL en lugar de los multiconjuntos.

22.5.1. Creación y acceso a los valores de los conjuntos

En SQL:1999 se puede crear un array de valores de esta manera:

array[´Silberschatz´, ´Korth´, ´Sudarshan´]

De manera parecida, se puede crear un multiconjunto de palabras clave de la manera siguiente:

multiset[´computadora´, ´base de datos´, ´SQL´]

Por tanto, se puede crear una tupla del tipo definido por la relación *libros* como:

(´Compiladores´, **array**[´Smith´, ´Jones´],
new *Editor*(´McGraw-Hill´, ´Nueva York´),
multiset[´análisis sintáctico´, ´análisis´])

En este ejemplo se ha creado un valor para el atributo *Editor* mediante la invocación a una función *constructora* para *Editor* con los argumentos correspondientes. Tenga en cuenta que esta función constructora para *Editor* se debe crear de manera explícita y no existe de manera predeterminada; se puede declarar como la constructora para *Nombre*, que ya se ha visto en la Sección 22.3.

Si se desea insertar la tupla anterior en la relación *libros*, se puede ejecutar la sentencia:

insert into *libros*
values
(´Compiladores´, **array**[´Smith´, ´Jones´],
new *Editor*(´McGraw-Hill´, ´Nueva York´),
multiset[´análisis sintáctico´, ´análisis´]);

Se puede tener acceso a los elementos del array o actualizarlos especificando el índice del array, por ejemplo, *array_autores* [1].

22.5.2. Consulta de los atributos valorados como conjuntos

Ahora se considerará la forma de manejar los atributos que se valoran como conjuntos. Las expresiones que se evalúan como conjuntos pueden aparecer en cualquier parte en la que pueda aparecer el nombre de una relación, como en las cláusulas **from,** como se muestra posteriormente. Se usará la tabla *libros* ya definida con anterioridad.

Si se desea encontrar todos los libros que tienen las palabras «base de datos» entre sus palabras clave se puede utilizar la siguiente consulta:

select *título*
from *libros*
where ´base de datos´ **in** (**unnest**(*palabras_clave*));

Observe que se ha usado **unnest**(*palabras_clave*) en una posición en la que SQL sin las relaciones anidadas habría exigido una subexpresión **select-from-where.**

Si se sabe que un libro concreto tiene tres autores, se puede escribir:

select *array_autores*[1], *array_autores*[2], *array_autores*[3]
from *libros*
where *título* = ´Fundamentos de bases de datos´;

Ahora suponga que se desea una relación que contenga parejas de la forma «título, nombre_autor» para cada libro y para cada uno de sus autores. Se puede usar esta consulta:

select *L.título*, *A.autores*
from *libros* **as** *L*, **unnest**(*L.array_autores*) **as** *A*(*autores*);

Dado que el atributo *array_autores* de *libros* es un campo como colección de valores, se puede usar **unnest**(*L.array_autores*) en una cláusula **from** en la que se espere una relación. Tenga en cuenta que la variable de tupla *L* es visible para esta expresión, ya que se ha definido *anteriormente* en la cláusula **from.**

Al desanidar un array la consulta anterior pierde información sobre el orden de los elementos del array. Se puede usar la cláusula **unnest with ordinality** para obtener esta información, como se muestra en la siguiente consulta. Esta consulta se puede usar para generar la relación *autores*, que se ha visto anteriormente, a partir de la relación *libros*.

select *título*, *A.autores*, *A.posición*
from *libros* **as** *L*,
unnest(*L.array_autores*)
with ordinality as *A*(*autores*, *posición*);

La cláusula **with ordinality** genera un atributo adicional que registra la posición del elemento en el array. Se puede utilizar una consulta parecida, pero sin la cláusula **with ordinality,** para generar la relación *palabras_clave*.

22.5.3. Anidamiento y desanidamiento

La transformación de una relación anidada en una forma con menos atributos de tipo relación (o sin ellos) se denomina **desanidamiento.** La relación *libros* tiene dos atributos, *array_autores* y *palabras_clave*, que son colecciones, y otros dos, *título* y *editor*, que no lo son. Suponga que se desea convertir la relación en una sola relación plana, sin relaciones anidadas ni tipos estructurados como atributos. Se puede usar la siguiente consulta para llevar a cabo la tarea:

select *título*, *A.autor*, *editor.nombre*
as *nombre_editor*, *editor.sucursal*
as *sucursal_editor*, *P.palabra_clave*
from *libros* **as** *L*, **unnest**(*L.array_autores*) **as** *A*(*autores*),
unnest (*L.palabras_clave*) **as** *P*(*palabra_clave*);

La variable *L* de la cláusula **from** se declara para que tome valores de *libros*. La variable *A* se declara para que tome valores de los autores de *array_autores* para el libro *L*, y *P* se declara para que tome valores de las palabras clave de *palabras_clave* del libro *L*. La Figura 22.1 muestra un ejemplar de la relación *libros* y la Figura 22.3 muestra la relación, denominada *libros_plana*, que es resultado de la consulta anterior. Tenga en cuenta que la relación *libros_plana* se halla en 1FN, ya que todos sus atributos son atómicos.

El proceso inverso de transformar una relación en la 1FN en una relación anidada se denomina **anidamiento.** El anidamiento puede realizarse mediante una extensión de la agrupación en SQL. En el

título	*autor*	*nombre_editor*	*sucursal_editor*	*palabras_clave*
Compiladores	Smith	McGraw-Hill	New York	análisis sintáctico
Compiladores	Jones	McGraw-Hill	New York	análisis sintáctico
Compiladores	Smith	McGraw-Hill	New York	análisis
Compiladores	Jones	McGraw-Hill	New York	análisis
Redes	Jones	Oxford	London	Internet
Redes	Frick	Oxford	London	Internet
Redes	Jones	Oxford	London	Web
Redes	Frick	Oxford	London	Web

Figura 22.3. *libros_plana*: resultado del desanidamiento de los atributos *array_autores* y *palabras_clave* de la relación *libros*.

título	*autor*	*editor*	*palabras_clave*
		(*nombre_editor*, *sucursal_editor*)	
Compiladores	Smith	(McGraw-Hill, New York)	{análisis sintáctico, análisis}
Compiladores	Jones	(McGraw-Hill, New York)	{análisis sintáctico, análisis }
Redes	Jones	(Oxford, London)	{Internet, web}
Redes	Frick	(Oxford, London)	{Internet, web}

Figura 22.4. Una versión parcialmente anidada de la relación *libros_plana*.

uso normal de la agrupación en SQL se crea (lógicamente) una relación multiconjunto temporal para cada grupo y se aplica una función de agregación a esa relación temporal para obtener un valor único (atómico). La función **collect** devuelve el multiconjunto de valores; en lugar de crear un solo valor se puede crear una relación anidada. Suponga que se tiene la relación en 1FN *libros_plana*, tal y como se muestra en la Figura 22.3. La consulta siguiente anida la relación en el atributo *palabras_clave*:

select *título*, *autores*, *Editor*(*nombre_editor*, *sucursal_editor*)
  as *editor*, **collect**(*palabras_clave*) **as** *palabras_clave*
from *libros_plana*
group by *título*, *autor*, *editor*;

El resultado de la consulta a la relación *libros_plana* de la Figura 22.3 aparece en la Figura 22.4.

Si se desea anidar también el atributo autores en un multiconjunto, se puede usar la consulta:

select *título*, **collect**(*autores*) **as** *conjunto_autores*,
  Editor(*nombre_editor*, *sucursal_editor*) **as** *editor*,
    collect(*palabra_clave*) **as** *palabras_clave*
from *libros_plana*
group by *título*, *editor*;

Otro enfoque de la creación de relaciones anidadas es usar subconsultas en la cláusula **select.** Una ventaja del enfoque de las subconsultas es que se puede usar de manera opcional en la subconsulta una cláusula **order by** para generar los resultados en el orden deseado, lo que puede aprovecharse para crear un array. La siguiente consulta ilustra este enfoque; las palabras clave **array** y **multiset** especifican que se van a crear un array y un multiconjunto (respectivamente) a partir del resultado de las subconsultas:

select *título*,
  array(**select** *autores*
    from *autores* **as** *A*
    where *A.título* = *L.título*
    order by *A.posición*) **as** *array_autores*,
  Editor(*nombre_editor*, *sucursal_editor*) **as** *editor*,
  multiset(**select** *palabra_clave*
    from *palabras_clave* **as** *P*
    where *P.título* = *L.título*) **as** *palabras_clave*,
from *libros4* **as** *L*;

El sistema ejecuta las subconsultas anidadas de la cláusula **select** para cada tupla generada por las cláusulas **from** y **where** de la consulta externa. Observe que el atributo *L.título* de la consulta externa se usa en las consultas anidadas para garantizar que solo se generen los conjuntos correctos de autores y de palabras clave para cada título.

SQL:2003 ofrece gran variedad de operadores para multiconjuntos, incluida la función **set**(*M*), que calcula una versión sin duplicados del multiconjunto *M*, la operación agregada **intersection,** cuyo resultado es la intersección de todos los multiconjuntos de un grupo, la operación agregada **fusion,** que devuelve la unión de todos los multiconjuntos de un grupo, y el predicado **submultiset,** que comprueba si el multiconjunto está contenido en otro multiconjunto.

La norma de SQL no proporciona ningún medio para actualizar los atributos de los multiconjuntos, salvo asignarles valores nuevos. Por ejemplo, para borrar el valor *v* del atributo de multiconjunto *A* hay que establecerlo como (*A* **except all multiset**[*v*]).

22.6. Identidad de los objetos y tipos de referencia en SQL

Los lenguajes orientados a objetos ofrecen la posibilidad de hacer referencia a objetos. Los atributos de un tipo dado pueden servir de referencia para los objetos de un tipo concreto. Por ejemplo, en SQL se puede definir el tipo *Departamento* con el campo *nombre* y el campo *director*, que es una referencia al tipo *Persona*, y la tabla *departamentos* del tipo *Departamento*, de la siguiente manera:

create type *Departamento* (
  nombre **varchar**(20),
  director **ref**(*Persona*) **scope** *personas*);

create table *departamentos* **of** *Departamento*;

En este caso, la referencia está restringida a las tuplas de la tabla *personas*. La restricción del **ámbito** (scope) de referencia a las tuplas de una tabla es obligatoria en SQL, y hace que las referencias se comporten como las claves externas.

Se puede omitir la declaración **scope** *personas* de la declaración de tipos y, en su lugar, añadir a la instrucción **create table**:

create table *departamentos* **of** *Departamento*
  (*director* **with options scope** *personas*);

La tabla a la que se hace referencia debe tener un atributo que guarde el identificador de cada tupla. Ese atributo, denominado **atributo autorreferencial** (selfreferential attribute), se declara añadiendo una cláusula **ref is** a la instrucción **create table:**

create table *personas* **of** *Persona*
ref is *id_personal* **system generated**;

En este caso, *id_personal* es el nombre de un atributo, no una palabra clave, y la sentencia **create table** especifica que la base de datos genera de manera automática el identificador.

Para inicializar el atributo de referencia hay que obtener el identificador de la tupla a la que se va a hacer referencia. Se puede conseguir el valor del identificador de la tupla mediante una consulta. Por tanto, para crear una tupla con el valor de referencia, primero se puede crear la tupla con una referencia nula y luego definir la referencia de manera independiente:

insert into *departamentos*
values (´CS´, null);

update *departamentos*
set *director* = (**select** *p.id_personal*
from *personal* **as** *p*
where *nombre* = ´John´)
where *nombre* = ´CS´;

Una alternativa a los identificadores generados por el sistema es permitir que los usuarios generen los identificadores. El tipo del atributo autorreferencial debe especificarse como parte de la definición de tipos de la tabla a la que se hace referencia, y la definición de la tabla debe especificar que la referencia está **generada por el usuario (user generated):**

create type *Persona*
(*nombre* **varchar**(20),
dirección **varchar**(20))
ref using varchar(20);

create table *personas* **of** *Persona*
ref is *id_personal* **user generated**;

Al insertar tuplas en *personas* hay que proporcionar el valor del identificador:

insert into *personas* (*id_personal*, *nombre*, *dirección*) **values**
(´01284567´, ´John´, ´23 Coyote Run´);

Ninguna otra tupla de *personas*, de sus supertablas ni de sus subtablas puede tener el mismo identificador.

Por tanto, se puede usar el valor del identificador al insertar tuplas en *departamentos*, sin necesidad de más consultas para recuperarlo:

insert into *departamentos*
values (´CS´, ´01284567´);

Incluso es posible usar el valor de una clave primaria ya existente como identificador, incluyendo la cláusula **ref from** en la definición de tipos:

create type *Persona*
(*nombre* **varchar**(20) **primary key**,
dirección **varchar**(20))
ref from(*nombre*);

create table *personas* **of** *Persona*
ref is *id_personal* **derived**;

Tenga en cuenta que la definición de la tabla debe especificar que las referencia es derivada, y debe seguir especificando el nombre de un atributo autorreferencial. Al insertar una tupla de *departamentos*, se puede usar:

insert into *departamentos*
values (´CS´, ´John´);

En SQL:1999 las referencias se desvinculan mediante el símbolo −>. Considere la tabla *departamentos* que se ha definido anteriormente. Se puede usar esta consulta para averiguar el nombre y la dirección de los directores de todos los departamentos:

select *director−>nombre*, *director−>dirección*
from *departamentos*;

Las expresiones como «*director−>nombre*» se denominan **expresiones de camino.**

Dado que *director* es una referencia a una tupla de la tabla *personas*, el atributo *nombre* de la consulta anterior es el atributo *nombre* de la tupla de la tabla *personas*. Se pueden usar las referencias para ocultar las operaciones de reunión; en el ejemplo anterior, sin las referencias, el campo *director* de *departamento* se declararía como clave externa de la tabla *personas*. Para averiguar el nombre y la dirección del director de un departamento hace falta una reunión explícita de las relaciones *departamentos* y *personas*. El uso de referencias simplifica considerablemente la consulta.

Se puede usar la operación **deref** para devolver la tupla a la que señala una referencia y luego tener acceso a sus atributos, como se muestra a continuación.

select deref(*director*).*nombre*
from *departamentos*;

22.7. Implementación de las características O-R

Los sistemas de bases de datos relacionales orientadas a objetos son básicamente extensiones de los sistemas de bases de datos relacionales ya existentes. Las modificaciones resultan claramente necesarias en muchos niveles del sistema de bases de datos. Sin embargo, para minimizar las modificaciones en el código del sistema de almacenamiento (almacenamiento de relaciones, índices, etc.), los tipos de datos complejos soportados por los sistemas relacionales orientados a objetos se pueden traducir al sistema de tipos más sencillo de las bases de datos relacionales.

Para comprender la manera de realizar esta traducción solo hace falta mirar al modo en que algunas características del modelo E-R se traducen en relaciones. Por ejemplo, los atributos multivalorados del modelo E-R se corresponden con los atributos valorados como multiconjuntos del modelo relacional orientado a objetos. Los atributos compuestos se corresponden aproximadamente con los tipos estructurados. Las jerarquías ES-UN del modelo E-R se corresponden con la herencia de tablas del modelo relacional orientado a objetos.

Las técnicas para convertir las características del modelo E-R en tablas, que se estudiaron en la Sección 7.6, se pueden usar, con algunas extensiones, para traducir los datos relacionales orientados a objetos a datos relacionales en el nivel de almacenamiento.

Las subtablas se pueden almacenar de manera eficiente, sin réplica de todos los campos heredados, de una de estas maneras:

- Cada tabla almacena la clave primaria (que puede haber heredado de una tabla madre) y los atributos que se definen localmente. No hace falta almacenar los atributos heredados (que no sean la clave primaria), se pueden obtener mediante una reunión con la supertabla, de acuerdo con la clave primaria.

- Cada tabla almacena todos los atributos heredados y definidos localmente. Cuando se inserta una tupla, solo se almacena en la tabla en la que se inserta, y su presencia se infiere en cada una de las supertablas. El acceso a todos los atributos de las tuplas es más rápido, ya que no hace falta ninguna reunión.

  No obstante, en caso de que el sistema de tipos permita que cada entidad se represente en dos subtablas, sin estar presente en una subtabla común a ambas, esta representación puede dar lugar a la réplica de información. Además, resulta difícil traducir las claves externas que hacen referencia a una supertabla en restricciones de las subtablas; para implementar de manera eficiente esas claves externas hay que definir la supertabla como vista, y el sistema de bases de datos tiene que soportar las claves externas en las vistas.

Las implementaciones pueden decidir representar los tipos arrays y multiconjuntos directamente o usar internamente una representación normalizada. Las representaciones normalizadas tienden a ocupar más espacio y exigen un coste adicional en reuniones o agrupamientos para reunir los datos en arrays o en multiconjuntos. Sin embargo, las representaciones normalizadas puede que sean más sencillas de implementar.

Las interfaces de programas de aplicación ODBC y JDBC se han extendido para recuperar y almacenar tipos estructurados; por ejemplo, JDBC ofrece el método getObject(), que es parecido a getString() pero devuelve un objeto Java Struct, a partir del cual se pueden extraer los componentes del tipo estructurado. También es posible asociar clases de Java con tipos estructurados de SQL, y JDBC puede realizar la conversión entre los tipos. Consulte el manual de referencia de ODBC o de JDBC para obtener más información.

22.8. Lenguajes de programación persistentes

Los lenguajes de las bases de datos se diferencian de los lenguajes de programación tradicionales en que trabajan directamente con datos que son persistentes; es decir, los datos siguen existiendo una vez que el programa que los creó haya concluido. Las relaciones de las bases de datos y las tuplas de las relaciones son ejemplos de datos persistentes. Por el contrario, los únicos datos persistentes con los que los lenguajes de programación tradicionales trabajan directamente son los archivos.

El acceso a las bases de datos es solo un componente de las aplicaciones del mundo real. Aunque los lenguajes para el tratamiento de datos como SQL son bastante efectivos en el acceso a los datos, se necesita un lenguaje de programación para implementar otros componentes de las aplicaciones como las interfaces de usuario o la comunicación con otras computadoras. La manera tradicional de realizar las interfaces de las bases de datos con los lenguajes de programación es incorporar SQL dentro del lenguaje de programación.

Los **lenguajes de programación persistentes** son lenguajes de programación extendidos con estructuras para el tratamiento de los datos persistentes. Los lenguajes de programación persistentes pueden distinguirse de los lenguajes con SQL incorporado, al menos, de dos formas:

1. Con los lenguajes incorporados, el sistema de tipos del lenguaje anfitrión suele ser diferente del sistema de tipos del lenguaje para el tratamiento de los datos. Los programadores son responsables de las conversiones de tipos entre el lenguaje anfitrión y SQL. Hacer que los programadores lleven a cabo esta tarea presenta varios inconvenientes:
   - El código para la conversión entre objetos y tuplas opera fuera del sistema de tipos orientado a objetos y, por tanto, tiene más posibilidades de presentar errores no detectados.
   - La conversión entre el formato orientado a objetos y el formato relacional de las tuplas en la base de datos necesita gran cantidad de código. El código para la conversión de formatos, junto con el código para cargar y descargar los datos de la base de datos, puede suponer un porcentaje significativo del código total necesario para la aplicación.

   Por el contrario, en los lenguajes de programación persistentes, el lenguaje de consultas se encuentra totalmente integrado con el lenguaje anfitrión y ambos comparten el mismo sistema de tipos. Los objetos se pueden crear y guardar en la base de datos sin ninguna modificación explícita del tipo o del formato; los cambios de formato necesarios se realizan de manera transparente.
2. Los programadores que usan lenguajes de consultas incorporados son responsables de la escritura de código explícito para la búsqueda en la memoria de los datos de la base de datos. Si se realizan actualizaciones, los programadores deben escribir explícitamente código para volver a guardar los datos actualizados en la base de datos.

   Por el contrario, en los lenguajes de programación persistentes, los programadores pueden manipular datos persistentes sin tener que escribir explícitamente código para buscarlos en la memoria o volver a guardarlos en el disco.

En esta sección se describe cómo se pueden extender los lenguajes de programación orientados a objetos, como C++ y Java, para hacerlos lenguajes de programación persistentes. Las características de estos lenguajes permiten que los programadores trabajen con los datos directamente desde el lenguaje de programación, sin tener que recurrir a lenguajes de tratamiento de datos como SQL. Por tanto, ofrecen una integración más estrecha de los lenguajes de programación con las bases de datos que, por ejemplo, SQL incorporado.

Sin embargo, los lenguajes de programación persistentes presentan ciertos inconvenientes que hay que tener presentes al decidir si utilizarlos o no. Dado que los lenguajes de programación suelen ser potentes, resulta relativamente sencillo cometer errores de programación que dañen las bases de datos. La complejidad de los lenguajes hace que la optimización automática de alto nivel, como la reducción de E/S de disco, resulte más difícil. En muchas aplicaciones, el soporte de las consultas declarativas resulta de gran importancia, pero actualmente los lenguajes de programación persistentes no soportan bien las consultas declarativas.

En esta sección se describen varios problemas teóricos que hay que abordar a la hora de añadir la persistencia a los lenguajes de programación ya existentes. En primer lugar se abordan los problemas independientes de los lenguajes y, en secciones posteriores, se tratan problemas específicos de los lenguajes C++ y Java. No obstante, no se tratan los detalles de las extensiones de los lenguajes; aunque se han propuesto varias normas, ninguna ha tenido una aceptación universal. Consulte las referencias de las notas bibliográficas para conocer más sobre extensiones de lenguajes concretas y más detalles de sus implementaciones.

22.8.1. Persistencia de los objetos

Los lenguajes de programación orientados a objetos ya poseen un concepto de objeto, un sistema de tipos para definir los tipos de los objetos y constructores para crearlos. Sin embargo, esos objetos son *transitorios*; desaparecen en cuanto finaliza el programa, igual que ocurre con las variables de los programas en Pascal o en C. Si se desea transformar uno de estos lenguajes en un lenguaje de programación de bases de datos, el primer paso consiste en pro-

porcionar una manera de hacer persistentes a los objetos. Se han propuesto varios enfoques:

- **Persistencia por clases.** El enfoque más sencillo, pero el menos conveniente, consiste en declarar que una clase es persistente. Todos los objetos de la clase son, por tanto, persistentes de manera predeterminada. Todos los objetos de las clases no persistentes son transitorios.

  Este enfoque no es flexible, dado que suele resultar útil disponer en una misma clase tanto de objetos transitorios como de objetos persistentes. Muchos sistemas de bases de datos orientados a objetos interpretan la declaración de que una clase es persistente como si se afirmara que los objetos de la clase pueden hacerse persistentes, en vez de que todos los objetos de la clase son persistentes. Estas clases se pueden denominar con más propiedad clases «que pueden ser persistentes».
- **Persistencia por creación.** En este enfoque se introduce una sintaxis nueva para crear los objetos persistentes mediante la extensión de la sintaxis para la creación de los objetos transitorios. Por tanto, los objetos son persistentes o transitorios en función de la forma de crearlos. Varios sistemas de bases de datos orientados a objetos siguen este enfoque.
- **Persistencia por marcas.** Una variante del enfoque anterior es marcar los objetos como persistentes después de haberlos creado. Todos los objetos se crean como transitorios pero, si un objeto tiene que persistir más allá de la ejecución del programa, hay que marcarlo como persistente de manera explícita antes de que este concluya. Este enfoque, a diferencia del anterior, pospone la decisión sobre la persistencia o la transitoriedad hasta después de la creación del objeto.
- **Persistencia por alcance.** Uno o varios objetos se declaran objetos persistentes (objetos raíz) de manera explícita. Todos los demás objetos serán persistentes si (y solo si) se pueden alcanzar desde algún objeto raíz mediante una secuencia de una o varias referencias.

  Por tanto, todos los objetos a los que se haga referencia desde (es decir, cuyos identificadores de objetos se guarden en) los objetos persistentes raíz serán persistentes. Pero también lo serán todos los objetos a los que se haga referencia desde ellos, y los objetos a los que estos últimos hagan referencia serán también persistentes, etc.

  Una ventaja de este esquema es que resulta sencillo hacer que sean persistentes estructuras de datos completas con solo declarar como persistente su raíz. Sin embargo, el sistema de bases de datos sufre la carga de tener que seguir las cadenas de referencias para detectar los objetos que son persistentes, y eso puede resultar costoso.

22.8.2. Identidad de los objetos y punteros

En los lenguajes de programación orientados a objetos que no se han extendido para tratar la persistencia, cuando se crea un objeto el sistema devuelve un identificador del objeto transitorio. Los identificadores de objetos transitorios solo son válidos mientras se ejecuta el programa que los ha creado; después de que concluya ese programa, el objeto se borra y el identificador pierde su sentido. Cuando se crea un objeto persistente, se le asigna un identificador de objeto persistente.

El concepto de identidad de los objetos tiene una relación interesante con los punteros de los lenguajes de programación. Una manera sencilla de conseguir una identidad intrínseca es usar los punteros a las ubicaciones físicas de almacenamiento. En concreto, en muchos lenguajes orientados a objetos, como C++, los identificadores de los objetos son en realidad punteros internos de la memoria.

Sin embargo, la asociación de los objetos con ubicaciones físicas de almacenamiento puede variar con el tiempo. Hay varios grados de permanencia de las identidades:

- **Dentro de los procedimientos.** La identidad solo persiste durante la ejecución de un único procedimiento. Un ejemplo de identidad dentro de los programas son las variables locales de los procedimientos.
- **Dentro de los programas.** La identidad solo persiste durante la ejecución de un único programa o de una única consulta. Un ejemplo de identidad dentro de los programas son las variables globales de los lenguajes de programación. Los punteros de la memoria principal o de la memoria virtual solo ofrecen identidad dentro de los programas.
- **Entre programas.** La identidad persiste de una ejecución del programa a otra. Los punteros a los datos del sistema de archivos del disco ofrecen identidad entre los programas, pero pueden cambiar si se modifica la manera en que los datos se guardan en el sistema de archivos.
- **Persistente.** La identidad no solo persiste entre las ejecuciones del programa sino también entre reorganizaciones estructurales de los datos. Es la forma persistente de identidad necesaria para los sistemas orientados a objetos.

En las extensiones persistentes de lenguajes como C++, los identificadores de objetos, de los objetos persistentes, se implementan como «punteros persistentes». Un *puntero persistente* es un tipo de puntero que, a diferencia de los punteros internos de la memoria, sigue siendo válido después del final de la ejecución del programa y después de algunas modalidades de reorganización de los datos. Los programadores pueden usar los punteros persistentes del mismo modo que usan los punteros internos de la memoria en los lenguajes de programación. Conceptualmente, los punteros persistentes se pueden considerar como punteros a objetos de la base de datos.

22.8.3. Almacenamiento y acceso a los objetos persistentes

¿Qué significa guardar un objeto en una base de datos? Evidentemente, hay que guardar por separado la parte de datos de cada objeto. Lógicamente, el código que implementa los métodos de las clases debe guardarse en la base de datos como parte de su esquema, junto con las definiciones de tipos de las clases. Sin embargo, muchas implementaciones se limitan a guardar el código en archivos externos a la base de datos para evitar tener que integrar el software del sistema, como los compiladores, con el sistema de bases de datos.

Hay varias maneras de encontrar los objetos de la base de datos. Una manera es dar nombres a los objetos, igual que se hace con los archivos. Este enfoque funciona con un número de objetos relativamente pequeño, pero no resulta práctico para millones de objetos. Una segunda manera es exponer los identificadores de los objetos o los punteros persistentes de los objetos, que pueden guardarse externamente. A diferencia de los nombres, los punteros no tienen por qué ser fáciles de recordar y pueden ser, incluso, punteros físicos internos de la base de datos.

Una tercera manera es guardar colecciones de objetos y permitir que los programas iteren sobre ellos para buscar los objetos deseados. Las colecciones de objetos pueden, a su vez, modelarse como objetos de un *tipo colección*. Entre los tipos de colecciones están los conjuntos, los multiconjuntos (es decir, conjuntos con varias apariciones posibles de un mismo valor), las listas, etc. Un caso especial de colección son las **extensiones de clases,** que son el conjunto de todos los objetos pertenecientes a una clase. Si hay una extensión de clase para una clase dada, siempre que se crea

un objeto de la clase ese objeto se inserta en la extensión de clase de manera automática; y, siempre que se borra un objeto, este se elimina de la extensión de clase. Las extensiones de clase permiten que las clases se traten como relaciones, en el sentido de que es posible examinar todos los objetos de una clase, igual que se pueden examinar todas las tuplas de una relación.

La mayor parte de los sistemas de bases de datos orientados a objetos admiten las tres maneras de acceso a los objetos persistentes. Dan identificadores a todos los objetos. Generalmente solo dan nombre a las extensiones de las clases y a otros objetos de tipo conjunto y, quizás, a otros objetos seleccionados, pero no a la mayor parte de los objetos. Las extensiones de las clases suelen conservarse para todas las clases que puedan tener objetos persistentes pero, en muchas de las implementaciones, las extensiones de las clases solo contienen los objetos persistentes de cada clase.

22.8.4. Sistemas persistentes de C++

Existen varias bases de datos orientadas a objetos basadas en las extensiones persistentes de C++ (consulte las notas bibliográficas). Hay diferencias entre ellas en términos de la arquitectura de los sistemas, pero tienen muchas características comunes en términos del lenguaje de programación.

Varias de las características orientadas a objetos del lenguaje C++ ayudan a proporcionar un buen soporte para la persistencia sin modificar el propio lenguaje. Por ejemplo, se puede declarar una clase denominada Persistent_Object (objeto persistente) con los atributos y los métodos para dar soporte a la persistencia; cualquier otra clase que deba ser persistente puede hacerse subclase de esta clase y heredará, por tanto, el soporte de la persistencia. El lenguaje C++ (igual que otros lenguajes modernos de programación) permite también redefinir los nombres de las funciones y los operadores estándar —como +, −, el operador de desvinculación de los punteros −>, etc.— en función del tipo de operandos a los que se aplican. Esta posibilidad se denomina *sobrecarga*; se usa para redefinir los operadores para que se comporten de la manera deseada cuando operan con objetos persistentes.

Proporcionar apoyo a la persistencia mediante las bibliotecas de clases presenta la ventaja de que solo se realizan cambios mínimos en C++; además, resulta relativamente fácil de implementar. Sin embargo, presenta el inconveniente de que los programadores tienen que utilizar mucho más tiempo para escribir los programas que trabajan con objetos persistentes y de que no les resulta sencillo especificar las restricciones de integridad del esquema ni ofrecer soporte para las consultas declarativas. Algunas implementaciones persistentes de C++ soportan extensiones de la sintaxis de C++ para facilitar estas tareas.

Es necesario abordar los siguientes problemas a la hora de añadir soporte a la persistencia a C++ (y a otros lenguajes):

- **Punteros persistentes.** Se debe definir un nuevo tipo de datos para que represente los punteros persistentes. Por ejemplo, la norma ODMG de C++ define la clase de plantillas d_Ref<*T*>, que representa punteros persistentes a la clase *T*. El operador de desvinculación para esta clase se vuelve a definir para que capture el objeto del disco (si no se halla ya presente en la memoria) y devuelva un puntero de la memoria al búfer en el que se ha capturado el objeto. Por tanto, si *p* es un puntero persistente a la clase *T*, se puede usar la sintaxis estándar como p−>A o p−>f(v) para tener acceso al atributo *A* de la clase *T* o para invocar al método *f* de la clase *T*.

  El sistema de bases de datos ObjectStore usa un enfoque diferente al de los punteros persistentes. Utiliza los tipos normales de punteros para almacenar los punteros persistentes. Esto plantea dos problemas: (1) el tamaño de los punteros de la memoria solo puede ser de 4 bytes, demasiado pequeño para las bases de datos mayores de 4 gigabytes, y (2) cuando se pasa algún objeto al disco, los punteros de la memoria que señalan a su antigua ubicación física carecen de significado. ObjectStore utiliza una técnica denominada «rescate hardware» para abordar ambos problemas; precaptura los objetos de la base de datos en la memoria y sustituye los punteros persistentes por punteros de memoria y, cuando se vuelven a almacenar los datos en el disco, los punteros de memoria se sustituyen por punteros persistentes. Cuando se hallan en el disco, el valor almacenado en el campo de los punteros de memoria no es el puntero persistente real; en vez de eso, se busca el valor en una tabla para averiguar el valor completo del puntero persistente.
- **Creación de objetos persistentes.** El operador new de C++ se usa para crear objetos persistentes mediante la definición de una versión «sobrecargada» del operador que usa argumentos adicionales especificando que deben crearse en la base de datos. Por tanto, en lugar de new T(), hay que llamar a new (bd) T() para crear un objeto persistente, donde bd identifica a la base de datos.
- **Extensiones de las clases.** Las extensiones de las clases se crean y se mantienen de manera automática para cada clase. La norma ODMG de C++ exige que el nombre de la clase se pase como parámetro adicional a la operación new. Esto también permite mantener varias extensiones para cada clase, pasando diferentes nombres.
- **Relaciones.** Las relaciones entre las clases se suelen representar almacenando punteros de cada objeto a los objetos con los que está relacionado. Los objetos relacionados con varios objetos de una clase dada almacenan un conjunto de punteros. Por tanto, si existen un par de objetos en una relación, cada uno de ellos debe almacenar un puntero al otro. Los sistemas persistentes de C++ ofrecen una manera de especificar esas restricciones de integridad y de hacer que se cumplan mediante la creación y borrado automático de los punteros. Por ejemplo, si se crea un puntero de un objeto *a* a un objeto *b*, se añade de manera automática un puntero a *a* en el objeto *b*.
- **Interfaz iteradora.** Dado que los programas tienen que iterar sobre los miembros de las clases, hace falta una interfaz para iterar sobre los miembros de las extensiones de las clases. La interfaz iteradora también permite especificar selecciones, de modo que solo hay que capturar los objetos que satisfagan el predicado de selección.
- **Transacciones.** Los sistemas persistentes de C++ ofrecen soporte para comenzar las transacciones y para comprometerlas o realizar un retroceso.
- **Actualizaciones.** Uno de los objetivos de ofrecer soporte a la persistencia a los lenguajes de programación es permitir la persistencia transparente. Es decir, una función que opere sobre un objeto no debe necesitar saber que el objeto es persistente; por tanto, se pueden usar las mismas funciones sobre los objetos independientemente de que sean persistentes o no.

  Sin embargo, surge el problema de que resulta difícil detectar cuándo se ha actualizado un objeto. Algunas extensiones persistentes de C++ exigían que el programador especificara de manera explícita que se había modificado el objeto llamando a la función mark_modified(). Además de incrementar el esfuerzo del programador, este enfoque aumenta la posibilidad de que se produzcan errores de programación que den lugar a una base de datos corrupta. Si un programador omite la llamada a mark_modified(), es posible que nunca se propague a la base de datos la actualización realizada por una transacción, mientras que otra actualización realizada por la misma transacción sí se propague, lo que viola la atomicidad de las transacciones.

Otros sistemas, como ObjectStore, usan soporte a la protección de la memoria proporcionada por el sistema operativo o por el hardware para detectar las operaciones de escritura en los bloques de memoria y marcar como sucios los bloques que se deban escribir posteriormente en el disco.

- **Lenguaje de consultas.** Los iteradores ofrecen soporte para consultas de selección sencillas. Para soportar consultas más complejas, los sistemas persistentes de C++ definen un lenguaje de consultas.

A finales de los años ochenta y principios de los noventa del siglo xx se desarrollaron un gran número de sistemas de bases de datos orientados a objetos basados en C++. Sin embargo, el mercado para esas bases de datos resultó mucho más pequeño de lo esperado, ya que la mayor parte de los requisitos de las aplicaciones se cumplen de sobra usando SQL mediante interfaces como ODBC o JDBC. En consecuencia, la mayor parte de los sistemas de bases de datos orientados a objetos desarrollados en ese periodo ya no existen. En los años noventa el Grupo de Gestión de Datos de Objetos (Object Data Management Group, ODMG) definió las normas para agregar persistencia a C++ y a Java. No obstante, el grupo concluyó sus actividades alrededor de 2002. ObjectStore y Versant son de los pocos sistemas de bases de datos orientados a objetos originales que siguen existiendo.

Aunque los sistemas de bases de datos orientados a objetos no encontraron el éxito comercial que esperaban, la razón de añadir persistencia a los lenguajes de programación sigue siendo válida. Hay varias aplicaciones con grandes exigencias de rendimiento que se ejecutan en sistemas de bases de datos orientados a objetos; el uso de SQL impondría una sobrecarga de rendimiento excesiva para muchos de esos sistemas. Con los sistemas de bases de datos relacionales orientados a objetos que proporcionan soporte para los tipos de datos complejos, incluidas las referencias, resulta más sencillo almacenar los objetos de los lenguajes de programación en bases de datos de SQL. Todavía puede emerger una nueva generación de sistemas de bases de datos orientados a objetos que utilicen bases de datos relacionales orientadas a objetos como sustrato.

22.8.5. Sistemas Java persistentes

En años recientes, el lenguaje Java ha visto un enorme crecimiento en su uso. La demanda de soporte de la persistencia de los datos en los programas de Java se ha incrementado de manera acorde. Los primeros intentos de creación de una norma para la persistencia en Java fueron liderados por el consorcio ODMG; posteriormente, el consorcio concluyó sus esfuerzos, pero transfirió su diseño al proyecto **Objetos de bases de datos de Java** (Java Database Objects, JDO), que coordina Sun Microsystems.

El modelo JDO para la persistencia de los objetos en los programas de Java es diferente del modelo de soporte de la persistencia en los programas de C++. Entre sus características se encuentran:

- **Persistencia por alcance.** Los objetos no se crean explícitamente en la base de datos. El registro explícito de un objeto como persistente (usando el método makePersistent() de la clase PersistenceManager) hace que el objeto sea persistente. Además, cualquier objeto alcanzable desde un objeto persistente pasa a ser persistente.
- **Mejora del código de bytes.** En lugar de declarar en el código de Java que una clase es persistente, se especifican en un archivo de configuración (con la extensión .jdo) las clases cuyos objetos se pueden hacer persistentes. Se ejecuta un programa *mejorador* específico de la implementación que lee el archivo de configuración y lleva a cabo dos tareas. En primer lugar, puede crear estructuras en la base de datos para almacenar objetos de esa clase. En segundo lugar, modifica el código de bytes (generado al compilar el programa de Java) para que maneje tareas relacionadas con la persistencia. A continuación se ofrecen algunos ejemplos de modificaciones de este tipo:
  - Se puede modificar cualquier código que tenga acceso a un objeto para que compruebe primero si el objeto se encuentra en la memoria y, si no está, dé los pasos necesarios para ponerlo en la memoria.
  - Cualquier código que modifique un objeto se modifica para que también registre el objeto como modificado y, quizás, para que guarde un valor previo a la actualización que se usa en caso de que haga falta deshacerla (es decir, si se retrocede la transacción).

  También se pueden llevar a cabo otras modificaciones del código de bytes. Esas modificaciones se pueden realizar porque el código de bytes es estándar en todas las plataformas e incluye mucha más información que el código objeto compilado.
- **Asignación de bases de datos.** JDO no define la manera en que se almacenan los datos en la base de datos subyacente. Por ejemplo, una situación frecuente es que los objetos se almacenen en una base de datos relacional. El programa mejorador puede crear en la base de datos un esquema adecuado para almacenar los objetos de las clases. La manera exacta en que lo hace depende de la implementación y no está definida por JDO. Se pueden asignar algunos atributos a los atributos relacionales, mientras que otros se pueden almacenar de forma serializada, que la base de datos trata como si fuera un objeto binario. Las implementaciones de JDO pueden permitir que los datos relacionales existentes se vean como objetos mediante la definición de la asignación correspondiente.
- **Extensiones de clase.** Las extensiones de clase se crean y se conservan de manera automática para cada clase declarada como persistente. Todos los objetos que se hacen persistentes se añaden de manera automática a la extensión de clase correspondiente a su clase. Los programas de JDO pueden tener acceso a las extensiones de clase e iterar sobre los miembros seleccionados. Se puede usar la interfaz Iterator de Java para crear iteradores sobre las extensiones de clase y avanzar por los miembros de cada extensión de clase. JDO también permite que se especifiquen selecciones cuando se crea una extensión de clase y que solo se capturen los objetos que satisfagan la selección.
- **Tipo de referencia único.** No hay diferencia de tipos entre las referencias a los objetos transitorios y las referencias a los objetos persistentes.

  Un enfoque para conseguir esta unificación de los tipos de puntero es cargar toda la base de datos en la memoria, sustituyendo todos los punteros persistentes por punteros en la memoria. Una vez llevadas a cabo las actualizaciones, el proceso se invierte y se vuelven a almacenar en el disco los objetos actualizados. Este enfoque es muy ineficiente para bases de datos de gran tamaño.

  A continuación se describirá un enfoque alternativo, que permite que los objetos persistentes se pasen a la memoria de manera automática cuando haga falta, a la vez que permite que todas las referencias contenidas en objetos de la memoria sean referencias de la memoria. Cuando se obtiene un objeto A, se crea un **objeto hueco** para cada objeto B_i al que haga referencia, y la copia de la memoria de A tiene referencias al objeto hueco correspondiente para cada B_i. Por supuesto, el sistema tiene que asegurar que, si ya se ha obtenido el objeto B_i, la referencia apunta al objeto ya obtenido en lugar de crear un nuevo objeto hueco. De manera parecida, si no se ha obtenido el objeto B_i, pero otro objeto ya obtenido hace referencia a él, ya tiene un objeto hueco creado para él; se usa la referencia al objeto hueco que ya existe, en lugar de crear un objeto hueco nuevo.

Por tanto, para cada objeto O_i que se haya obtenido, cada referencia de O_i es a un objeto ya capturado o a un objeto hueco. Los objetos huecos forman un *borde* que rodea los objetos obtenidos.

Siempre que el programa tiene acceso real al objeto hueco O, el código de bytes mejorado lo detecta y obtiene el objeto de la base de datos. Cuando se consigue este objeto, se lleva a cabo el mismo proceso de creación de objetos huecos para todos los objetos a los que O hace referencia. Tras esto, se permite que se lleve a cabo el acceso al objeto.[5]

Para implementar este esquema hace falta una estructura de índices en la memoria que asigne los punteros persistentes a las referencias de la memoria. Cuando se vuelven a escribir los objetos en el disco, se usa ese índice para sustituir las referencias de la memoria por punteros persistentes en la copia que se escribe en el disco.

22.9. Correspondencia entre el modelo relacional y el orientado a objetos

Hasta aquí se han visto dos enfoques para integrar los modelos de datos orientados a objetos y los lenguajes de programación con las bases de datos. Los sistemas de **correspondencia** entre modelos **relacional** y **orientado a objetos** proporcionan un tercer enfoque de integración entre los lenguajes de programación orientados a objetos y las bases de datos.

Los sistemas de correspondencia relacional-orientado a objetos se construyen sobre las bases de datos relacionales tradicionales y permiten que un programador defina una correspondencia entre las tuplas de las relaciones de la base de datos y los objetos del lenguaje de programación. Al contrario que en los lenguajes de programación persistentes, los objetos son temporales y no existe una identidad de objeto permanente.

Se puede obtener un objeto, o un conjunto de objetos, a partir de una condición de selección sobre sus atributos; los datos relevantes se obtienen de la base de datos subyacente de acuerdo con las condiciones de la selección y se crean uno o más objetos a partir de los datos obtenidos, de acuerdo con la correspondencia preestablecida entre los objetos y las relaciones. El programa puede, opcionalmente, actualizar dichos objetos, crear otros nuevos o borrar algunos y, después, realizar un comando de guardar; se usa entonces la correspondencia entre los objetos y las relaciones para llevar a cabo la correspondiente actualización, inserción o borrado de tuplas en la base de datos.

Los sistemas de correspondencia entre modelos relacional y orientado a objetos en general, y en particular el sistema Hibernate, que proporciona una correspondencia entre Java y una base de datos relacional, se describen con detalle en la Sección 9.4.2.

El principal objetivo de los sistemas de correspondencia entre modelos relacional y orientado a objetos es facilitar al programador la tarea de construir aplicaciones, proporcionando un modelo de objetos, a la vez que mantienen las ventajas de utilizar por debajo una base de datos relacional robusta. Además, con la ventaja adicional de que cuando opera con objetos en memoria caché, los sistemas relacional-orientado a objetos pueden proporcionar una significativa mejora en el rendimiento sobre el acceso directo a la base de datos.

Los sistemas de correspondencia entre modelos relacional y orientado a objetos también proporcionan lenguajes de consulta que permiten que un programador pueda escribir directamente consultas sobre el modelo de objetos; estas consultas se traducen a consultas de SQL en la base de datos relacional y los objetos obtenidos se crean a partir de los resultados de la consulta en SQL.

En el lado negativo, estos sistemas pueden sufrir de una sobrecarga excesiva de actualizaciones masivas de la base de datos y proporcionar solo unas capacidades de consulta limitadas. Sin embargo, se puede actualizar directamente la base de datos, sin usar el sistema de correspondencia entre modelos relacional y orientado a objetos, y escribir consultas complejas directamente en SQL. Las ventajas de estos sistemas superan sus desventajas para muchas aplicaciones y se han extendido ampliamente en los últimos años.

22.10. Sistemas orientados a objetos y sistemas relacionales orientados a objetos

Ya se han estudiado las bases de datos relacionales orientadas a objetos, que son bases de datos orientadas a objetos construidas sobre el modelo relacional, así como las bases de datos orientadas a objetos, que se crean alrededor de los lenguajes de programación persistentes, y los sistemas de asociación objetos-relacional, que construyen una capa de objetos sobre una base de datos relacional tradicional.

Cada uno de estos enfoques se dirige a mercados diferentes. La naturaleza declarativa y la limitada potencia (comparada con la de los lenguajes de programación) del lenguaje SQL proporcionan una buena protección de los datos respecto a los errores de programación y hacen que las optimizaciones de alto nivel, como la reducción de E/S, resulten relativamente sencillas (la optimización de las expresiones relacionales se trató en el Capítulo 13). Los sistemas relacionales orientados a objetos se dirigen a simplificar la realización de los modelos de datos y de las consultas mediante el uso de tipos de datos complejos. Entre las aplicaciones habituales están el almacenamiento y la consulta de datos complejos, incluidos los datos multimedia.

Un lenguaje declarativo como SQL, sin embargo, impone una reducción significativa del rendimiento a ciertos tipos de aplicaciones que se ejecutan principalmente en la memoria principal y realizan gran número de accesos a la base de datos. Los lenguajes de programación persistentes se dirigen a las aplicaciones de este tipo que tienen necesidad de un rendimiento elevado. Proporcionan acceso a los datos persistentes con poca sobrecarga y eliminan la necesidad de traducir los datos si hay que tratarlos con un lenguaje de programación. Sin embargo, son más susceptibles de deteriorar los datos por errores de programación y no suelen disponer de gran capacidad de consulta. Entre las aplicaciones habituales están las bases de datos de CAD.

Los sistemas de correspondencia entre modelos relacional y orientado a objetos permiten que los programadores construyan aplicaciones usando un modelo de objetos, al mismo tiempo que utilizan un sistema de bases de datos tradicional para almacenar los datos. Por tanto, combinan la robustez de los sistemas de bases de datos relacionales de gran uso con la potencia de los modelos de objetos para escribir las aplicaciones. Sin embargo, sufren la sobrecarga de la conversión entre el modelo de objetos y el modelo relacional utilizado para guardar los datos.

5 La técnica que usa objetos huecos, descrita anteriormente, se encuentra estrechamente relacionada con la técnica de rescate hardware (ya mencionada en la Sección 22.8.4). El rescate hardware es utilizado por algunas implementaciones persistentes de C++ para ofrecer un solo tipo de puntero para los punteros persistentes y los de memoria. El rescate hardware utiliza técnicas de protección de la memoria virtual proporcionadas por el sistema operativo para detectar el acceso a las páginas y obtiene las páginas de la base de datos cuando es necesario. Por el contrario, la versión de Java modifica el código de bytes para que compruebe la existencia de objetos huecos, en lugar de usar la protección de la memoria, y obtiene los objetos cuando hace falta, en vez de obtener páginas enteras de la base de datos.

Los puntos fuertes de los diversos tipos de sistemas de bases de datos se pueden resumir de la siguiente forma:

- **Sistemas relacionales:** tipos de datos sencillos, lenguajes de consultas potentes, protección elevada.
- **Bases de datos orientadas a objetos (OOBD) basadas en lenguajes de programación persistentes:** tipos de datos complejos, integración con los lenguajes de programación, elevado rendimiento.
- **Sistemas relacionales orientados a objetos:** tipos de datos complejos, lenguajes de consulta potentes, protección elevada.
- **Sistemas de correspondencia** entre modelos **relacional** y **orientado a objetos:** tipos de datos complejos integrados con los lenguajes de programación; se diseñan como una capa por encima del sistema de bases de datos.

Estas descripciones son válidas en general, pero hay que tener en cuenta que algunos sistemas de bases de datos no respetan estas fronteras. Por ejemplo, algunos sistemas de bases de datos orientados a objetos construidos alrededor de lenguajes de programación persistentes se pueden implementar sobre sistemas de bases de datos relacionales o sobre sistemas de bases de datos relacionales orientados a objetos. Puede que estos sistemas proporcionen menor rendimiento que los sistemas de bases de datos orientados a objetos construidos directamente sobre los sistemas de almacenamiento, pero proporcionan parte de las garantías de protección más estrictas propias de los sistemas relacionales.

22.11. Resumen

- El modelo de datos relacional orientado a objetos extiende el modelo de datos relacional al proporcionar un sistema de tipos enriquecido que incluye los tipos de conjuntos y la orientación a objetos.
- Los tipos colección incluyen las relaciones anidadas, los conjuntos, los multiconjuntos y los arrays, y el modelo relacional orientado a objetos permite que los atributos de las tablas sean colecciones.
- La orientación a objetos proporciona herencia con subtipos y subtablas, así como referencias a objetos (tuplas).
- La norma SQL extiende el lenguaje de definición de datos y de consultas con los nuevos tipos de datos y la orientación a objetos. Incluyen el soporte a los atributos valorados como conjuntos, a la herencia y a las referencias a las tuplas. Estas extensiones intentan conservar los fundamentos relacionales —en particular, el acceso declarativo a los datos— a la vez que se extiende la potencia de modelado.
- Los sistemas de bases de datos relacionales orientados a objetos (es decir, los sistemas de bases de datos basados en el modelo relacional orientado a objetos) ofrecen un camino de migración cómodo para los usuarios de las bases de datos relacionales que desean usar las características orientadas a objetos.
- Las extensiones persistentes de C++ y Java integran la persistencia de forma elegante y ortogonalmente a sus elementos de programación previos, por lo que resultan fáciles de usar.
- La norma ODMG define las clases y otros constructores para la creación y acceso a los objetos persistentes desde C++, mientras que la norma JDO ofrece una funcionalidad equivalente para Java.
- Los sistemas de correspondencia entre modelos relacional y orientado a objetos proporcionan una vista de los datos almacenados en la base de datos relacional. Los objetos son temporales y no existe la noción de identidad de objetos persistentes. Los objetos se crean bajo demanda a partir de los datos relacionales y las actualizaciones de objetos se implementan como actualizaciones de los datos relacionales. Estos sistemas se han adoptado extensamente, al contrario que la adopción más limitada de los lenguajes de programación persistentes.
- Se han estudiado las diferencias entre los lenguajes de programación persistentes y los sistemas relacionales orientados a objetos, y se han mencionado criterios para escoger entre ellos.

Términos de repaso

- Relaciones anidadas.
- Modelo relacional anidado.
- Tipos complejos.
- Tipos colección.
- Tipos de objetos grandes.
- Conjuntos.
- Arrays.
- Multiconjuntos.
- Tipos estructurados.
- Métodos.
- Tipos de fila.
- Constructores.
- Herencia.
  - Simple.
  - Múltiple.
- Herencia de tipos.
- Tipo más concreto.
- Herencia de tablas.
- Subtabla.
- Solapamiento de subtablas.
- Tipos de referencia.
- Ámbito de una referencia.
- Atributo autorreferencial.
- Expresiones de camino.
- Anidamiento y desanidamiento.
- Funciones y procedimientos en SQL.
- Lenguajes de programación persistentes.
- Persistencia por:
  - Clase.
  - Creación.
  - Marcado.
  - Alcance.
- Enlace C++ de ODMG.
- ObjectStore.
- JDO.
  - Persistencia por alcance.
  - Raíces.
  - Objetos huecos.
- Asignación relacional de objetos.

Ejercicios prácticos

22.1. Una compañía de alquiler de coches tiene una base de datos con todos los vehículos de su flota actual. Para todos los vehículos incluye el número de bastidor, el número de matrícula, el fabricante, el modelo, la fecha de adquisición y el color. Se incluyen datos especiales para algunos tipos de vehículos:

- Camiones: capacidad de carga.
- Coches deportivos: potencia, edad mínima del arrendatario.
- Furgonetas: número de plazas.
- Vehículos todoterreno: altura de los bajos, eje motor (tracción a dos ruedas o a las cuatro).

Construya una definición del esquema de esta base de datos de acuerdo con SQL. Utilice la herencia donde resulte conveniente.

22.2. Considere el esquema de la base de datos con la relación *Emp* cuyos atributos se muestran a continuación, con los tipos especificados para los atributos multivalorados.

Emp = *(nombre, ConjuntoHijos* **multiset***(HijosC), ConjuntoConocimientos* **multiset***(Conocimientos))*
Hijos = (*nombre, cumpleaños*)
Conocimientos = *(mecanografía, ConjuntoExámenes* **setof***(Exámenes))*
Exámenes = *(año, ciudad)*

a. Defina el esquema anterior en SQL, con los tipos apropiados para cada atributo.

b. Usando el esquema anterior, escriba las consultas siguientes en SQL:

i Encontrar el nombre de todos los empleados que tienen un hijo nacido el 1 de enero de 2000 o en fechas posteriores.

ii Encontrar los empleados que han hecho un examen del tipo de conocimiento «mecanografía» en la ciudad de «Dayton».

iii Hacer una relación de los tipos de conocimiento de la relación *Emp*.

22.3. Considere el diagrama E-R de la Figura 22.5, que contiene atributos compuestos, multivalorados y derivados.

a. Indique una definición de esquema en SQL:2003 correspondiente al diagrama E-R.

b. Indique constructores para cada uno de los tipos estructurados definidos.

22.4. Considere el esquema relacional de la Figura 22.6.

a. Indique una definición de esquema en SQL correspondiente al esquema relacional, pero usando referencias para expresar las relaciones de clave externa.

b. Escriba cada una de las consultas del Ejercicio 6.13 sobre el esquema anterior usando SQL.

22.5. Suponga que trabaja como asesor para escoger un sistema de bases de datos para la aplicación del cliente. Para cada una de las aplicaciones siguientes, indique el tipo de sistema de bases de datos (relacional, base de datos orientada a objetos basada en un lenguaje de programación persistente, relacional orientada a objetos; no se debe especificar ningún producto comercial) que recomendaría. Justifique su recomendación.

a. Sistema de diseño asistido por computadora para un fabricante de aviones.

b. Sistema para realizar el seguimiento de los donativos hechos a los candidatos a un cargo público.

c. Sistema de información de ayuda para la realización de películas.

22.6. ¿En qué se diferencia el concepto de objeto del modelo orientado a objetos del concepto de entidad del modelo entidad-relación?

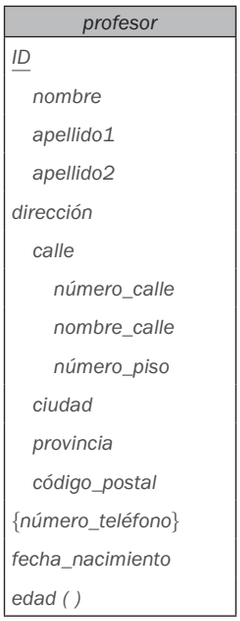


Figura 22.5. Diagrama E-R que contiene atributos compuestos, multivalorados y derivados.

empleado (*nombre_empleado*, *calle*, *ciudad*)
trabaja (*nombre_empleado*, *nombre_empresa*, *sueldo*)
empresa (*nombre_empresa*, *ciudad*)
supervisa (*nombre_empleado*, *nombre_jefe*)

Figura 22.6. Base de datos relacional para el Ejercicio práctico 22.4.

Ejercicios

22.7. Vuelva a diseñar la base de datos del Ejercicio práctico 22.2 en la primera y en la cuarta formas normales. Indique las dependencias funcionales o multivaloradas que se den por supuestas. Indique también todas las restricciones de integridad referencial que deban incluirse en los esquemas de la primera y de la cuarta formas normales.

22.8. Considere el esquema del Ejercicio práctico 22.2.

a. Escriba las instrucciones del LDD de SQL para crear la relación *EmpA*, que tiene la misma información que *Emp*, pero en la que los atributos valorados como multiconjuntos *ConjuntoNiños*, *ConjuntoConocimientos* y *ConjuntoExámenes* se sustituyen por los atributos valorados como arrays *ArrayHijos*, *ArrayConocimientos* y *ArrayExámenes*.

b. Escriba una consulta para transformar los datos del esquema de *Emp* al de *EmpA*, con el array de los hijos ordenado por fecha de cumpleaños, el de conocimientos por el tipo de conocimientos y el de exámenes por el año.

c. Escriba una sentencia de SQL para actualizar la relación *Emp* añadiendo el hijo John, con fecha de nacimiento 5 de febrero de 2001, al empleado llamado George.

d. Escriba una sentencia de SQL para llevar a cabo la misma actualización que antes, pero sobre la relación *EmpA*. Asegúrese de que el array de los hijos sigue ordenado por años.

22.9. Considere los esquemas de la tabla *personas* y las tablas *estudiantes* y *profesores* que se crearon bajo *personas* en la Sección 22.4. Indique un esquema relacional en la tercera forma normal que represente la misma información. Recuerde las restricciones de las subtablas y escriba todas las restricciones que deban imponerse en el esquema relacional para que cada ejemplar de la base de datos del esquema relacional pueda representarse también mediante un ejemplar del esquema con herencia.

22.10. Explique la diferencia entre el tipo x y el tipo de referencia **ref**(x). ¿En qué circunstancias se debe escoger usar el tipo de referencia?

22.11. Considere el diagrama E-R de la Figura 22.7, que contiene especializaciones, usando subtipos y subtablas.

a. Indique una definición del esquema SQL del diagrama E-R.

b. Escriba una consulta de SQL para encontrar el nombre de todas las personas que no son secretarias.

c. Escriba una consulta de SQL para imprimir el nombre de las personas que no sean empleados ni clientes.

d. ¿Se puede crear una persona que sea a la vez empleado y cliente con el esquema que se acaba de crear? Explique la manera de hacerlo o el motivo por el que no es posible.

22.12. Suponga que una base de datos de JDO tiene un objeto A, que hace referencia al objeto B, que, a su vez, hace referencia al objeto C. Suponga que todos los objetos se hallan inicialmente en el disco. Suponga que un programa desvincula en primer lugar a A, luego a B siguiendo la referencia de A y, finalmente, desvincula a C. Indique los objetos que están representados en la memoria tras cada desvinculación, junto con su estado (hueco o lleno, y los valores de los campos de referencia).

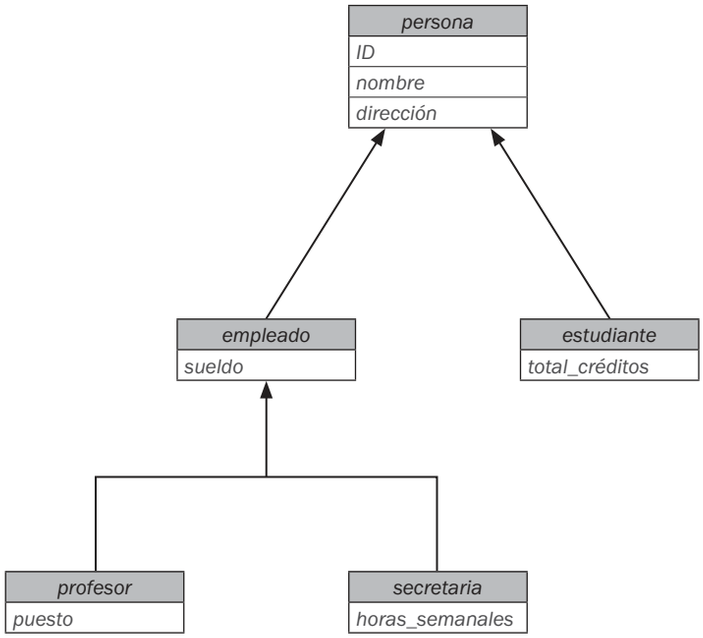


Figura 22.7. Especialización y generalización.

Herramientas

Existen considerables diferencias entre los diversos productos de bases de datos en cuanto al soporte de las características relacionales orientadas a objetos. Probablemente Oracle tenga el soporte más amplio entre los principales fabricantes de bases de datos. El sistema de bases de datos de Informix ofrece soporte para muchas características relacionales orientadas a objetos. Tanto Oracle como Informix ofrecían características relacionales orientadas a objetos antes de la finalización de la norma SQL:1999 y presentan algunas características que no forman parte de SQL:1999.

La información sobre ObjectStore y Versant, incluida la descarga de versiones de prueba, se puede obtener de sus respectivos sitios web (objectstore.com y versant.com). El proyecto Apache DB (db.apache.org) ofrece una herramienta de asociación relacional orientada a objetos para Java que soporta tanto Java de ODMG como las API de JDO. Se puede obtener una implementación de referencia de JDO de sun.com; use un motor de búsqueda para obtener la URL completa.

Notas bibliográficas

Se han propuesto varias extensiones de SQL orientadas a objetos. POSTGRES (Stonebraker y Rowe [1986] y Stonebraker [1986]) fue una de las primeras implementaciones de un sistema relacional orientado a objetos. Otros sistemas relacionales orientados a objetos de la primera época son las extensiones de SQL de O_2 (Bancilhon et ál. [1989]) y UniSQL (UniSQL [1991]). SQL:1999 fue producto de un amplio (y muy tardío) esfuerzo de normalización, que se inició originalmente mediante el añadido a SQL de características orientadas a objetos y acabó añadiendo muchas más características, como las estructuras procedimentales que se han visto anteriormente. El soporte de los tipos multiconjuntos se añadió como parte de SQL:2003.

Melton [2002] se centra en las características relacionales orientadas a objetos de SQL:1999. Eisenberg et ál. [2004] ofrecen una visión general de SQL:2003, incluido el soporte de los multiconjuntos.

A finales de los años ochenta y principios de los años noventa del siglo XX se desarrollaron varios sistemas de bases de datos orientadas a objetos. Entre los más notables de los comerciales figuran ObjectStore (Lamb et ál. [1991]), O_2 (Lecluse et ál. [1988]) y Versant. La norma para bases de datos orientadas a objetos ODMG se describe con detalle en Cattell [2000]. JDO se describe en Roos [2002], Tyagi et ál. [2003] y Jordan y Russell [2003].

XML

23

El **lenguaje de marcas extensible** (Extensible Markup Language, XML) no se concibió como una tecnología para bases de datos. En realidad, al igual que el *lenguaje de marcas de hipertexto* (*HyperText Markup Language*, HTML) en el que está basado la World Wide Web, XML tiene sus raíces en la gestión de documentos y deriva de un lenguaje para estructurar documentos grandes conocido como *lenguaje estándar generalizado de marcas* (*Standard Generalized Markup Language*, SGML). Sin embargo, a diferencia de SGML y de HTML, XML se diseñó para representar datos. Es particularmente útil como formato de datos cuando las aplicaciones se deben comunicar con otra aplicación o integrar información de varias aplicaciones. Cuando se utiliza XML en estos contextos, se generan muchas dudas sobre las bases de datos, incluyendo cómo organizar, manipular y consultar los datos XML. En este capítulo se presenta XML y se estudia la gestión de los datos XML con las técnicas de bases de datos, así como el intercambio de datos con formato documentos XML.

23.1. Motivación

Para comprender XML es importante entender sus raíces como un lenguaje de marcas de documentos. El término **marca** se refiere a cualquier elemento de un documento que no se tiene intención de que forme parte de la salida impresa. Por ejemplo, un escritor que crea un texto que finalmente se compone en una revista puede desear realizar notas sobre cómo se ha de realizar la composición. Sería importante introducir estas notas de tal forma que se pudieran distinguir del contenido real; una nota como «usar un tipo mayor y poner en negrita» o «no romper este párrafo» no acabaría impresa en la revista. Estas notas comunican información adicional sobre el texto. En un procesamiento electrónico de documentos, un **lenguaje de marcas** es una descripción formal de la parte del documento que es contenido, la parte que es marca y lo que significa la marca.

Así como los sistemas de bases de datos evolucionaron desde el procesamiento físico de archivos para proporcionar una vista lógica aislada, los lenguajes de marcas evolucionaron desde la especificación de instrucciones que indicaban cómo imprimir partes del documento para la *función* del contenido. Por ejemplo, con marcas funcionales, el texto que representa los encabezamientos de sección (para esta sección, la palabra «Motivación») se marcaría como un encabezamiento de sección en lugar de marcarse como un texto con el fin de imprimirse en tamaño grande y negrita. Desde el punto de vista del editor, dicha marca funcional permite que el documento tenga distintos formatos en contextos diferentes. También ayuda a que distintas partes de un documento largo, o distintas páginas de un sitio web grande, tengan un formato uniforme. Más importante, la marca funcional también ayuda a registrar lo que representa semánticamente cada parte del texto y ayuda a la extracción automática de partes claves de los documentos.

Para la familia de lenguajes de marcado, en los que se incluyen HTML, SGML y XML, las marcas adoptan la forma de **etiquetas** encerradas entre corchetes angulares, <>. Las etiquetas se usan en pares, con <etiqueta> y </etiqueta> delimitando el comienzo y el final de la parte del documento a la cual se refiere la etiqueta. Por ejemplo, el título de un documento podría estar marcado de la siguiente forma:

<título>Fundamentos de bases de datos</título>

A diferencia de HTML, XML no impone las etiquetas permitidas, y se pueden elegir como sea necesario para cada aplicación. Esta característica es la clave de la función principal de XML en la representación e intercambio de datos, mientras que HTML se usa principalmente para dar formato a los documentos.

Por ejemplo, en nuestra aplicación de la universidad, la información de departamento, asignatura y profesor se puede representar como parte de un documento XML, como se muestra en la Figura 23.1. Observe el uso de etiquetas tales como departamento, asignatura, profesor y enseña. Para hacer más sencillo el ejemplo se ha usado una versión simplificada de la universidad que ignora la información de secciones para las asignaturas. También se ha utilizado la etiqueta IID para indicar el identificador del profesor, por razones que veremos más adelante.

Estas etiquetas proporcionan el contexto de cada valor y permiten identificar la semántica del valor. Para este ejemplo, la representación de datos XML no proporciona ninguna ventaja significativa con respecto a la representación relacional; sin embargo, se usa este ejemplo por su simplicidad.

La Figura 23.2 muestra cómo se puede representar un pedido de compra en XML, que ilustra un uso más realista de XML. Los pedidos de compra habitualmente se generan por una empresa y se envían a otra. Tradicionalmente el comprador los imprimía en papel y los enviaba al proveedor; este reintroducía manualmente los datos en el sistema informático. Este lento proceso se puede acelerar en gran medida enviando electrónicamente la información entre el comprador y el proveedor. La representación anidada permite que

toda la información de un pedido de compra se represente de forma natural en un único documento (los pedidos de compra reales tienen considerablemente más información que la que se muestra en este sencillo ejemplo). XML proporciona una forma estándar de etiquetar los datos; obviamente, las dos empresas deben estar de acuerdo en las etiquetas que aparecen en el pedido de compra y en lo que significan.

Comparado con el almacenamiento de los datos en una base de datos relacional, la representación XML puede parecer poco eficiente, puesto que los nombres de las etiquetas se repiten por todo el documento. Sin embargo, a pesar de esta desventaja, una representación XML presenta ventajas significativas cuando se usa para el intercambio de datos entre empresas y para almacenar información estructurada en archivos:

- En primer lugar, la presencia de las etiquetas hace que el mensaje sea **autodocumentado;** es decir, no se tiene que consultar un esquema para comprender el significado del texto. Por ejemplo, se puede leer fácilmente el fragmento anterior.
- En segundo lugar, el formato del documento no es rígido. Por ejemplo, si algún remitente agrega información adicional tal como una etiqueta último_acceso que informa de la última fecha en la que se ha accedido a la cuenta, el receptor de los datos XML puede sencillamente ignorar la etiqueta. Como ejemplo adicional, en la Figura 23.2 el elemento con el identificador SG2 tiene una etiqueta denominada unidad_de_medida, que el primer elemento no tiene. La etiqueta es necesaria en los elementos que se ordenan por peso o volumen y se puede omitir en aquellos que simplemente se ordenan por número.

  La capacidad de reconocer e ignorar las etiquetas inesperadas permite al formato de los datos evolucionar con el tiempo sin invalidar las aplicaciones existentes. De forma similar, la posibilidad de que la misma etiqueta aparezca varias veces hace que sea fácil representar atributos multivalorados.
- En tercer lugar, XML permite estructuras anidadas. El pedido de compra que se muestra en la Figura 23.2 muestra las ventajas de tener estas estructuras. Cada pedido de compra tiene un comprador y una lista de elementos como dos de sus estructuras anidadas. Cada elemento tiene a su vez un identificador, una descripción y un precio anidados, mientras que el comprador tiene anidados un nombre y una dirección.

  En el modelo relacional esta información se podría haber dividido en varias relaciones. La información de los elementos se habría almacenado en una relación, la información del comprador en una segunda relación, los pedidos de compra en una tercera y la relación entre los pedidos de compra, los compradores y los elementos en una cuarta.

  La representación relacional ayuda a evitar la redundancia; por ejemplo, en un esquema relacional normalizado las descripciones de los elementos solo se almacenarían una vez por cada identificador de elemento. Sin embargo, en el pedido de compra de XML, las descripciones se pueden repetir en varios pedidos con el mismo elemento. No obstante, es atractivo recoger toda la información relacionada con un pedido en una única estructura anidada, incluso añadiendo redundancia, cuando la información se tiene que intercambiar con terceros.
- Finalmente, puesto que el formato XML está ampliamente aceptado, hay una gran variedad de herramientas disponibles para ayudar a su procesamiento, incluyendo API para lenguajes de programación para crear y leer datos XML, software de exploración y herramientas de bases de datos.

Más adelante, en la Sección 23.7, se describen varias aplicaciones de XML. Al igual que SQL es el *lenguaje* dominante para consultar los datos relacionales, XML se está convirtiendo en el *formato* dominante para el intercambio de datos.

```
<universidad>
  <departamento>
    <nombre_dept> Informática </nombre_dept>
    <edificio> Taylor </edificio>
    <presupuesto> 100000 </presupuesto>
  </departamento>
  <departamento>
    <nombre_dept> Biología </nombre_dept>
    <edificio> Watson </edificio>
    <presupuesto> 90000 </presupuesto>
  </departamento>
  <asignatura>
    <asignatura_id> CS-101 </asignatura_id>
    <nombre_asig> Intro. a la Informática </nombre_asig>
    <nombre_dept> Informática </nombre_dept>
    <créditos> 4 </créditos>
  </asignatura>
  <asignatura>
    <asignatura_id> BIO-301 </asignatura_id>
    <nombre_asig> Genética </nombre_asig>
    <nombre_dept> Biología </nombre_dept>
    <créditos> 4 </créditos>
  </asignatura>
  <profesor>
    <IID> 10101 </IID>
    <nombre> Srinivasan </nombre>
    <nombre_dept> Informática </nombre_dept>
    <sueldo> 65000 </sueldo>
  </profesor>
  <profesor>
    <IID> 83821 </IID>
    <nombre> Brandt </nombre>
    <nombre_dept> Informática </nombre_dept>
    <sueldo> 92000 </sueldo>
  </profesor>
  <profesor>
    <IID> 76766 </IID>
    <nombre> Crick </nombre>
    <nombre_dept> Biología </nombre_dept>
    <sueldo> 72000 </sueldo>
  </profesor>
  <enseña>
    <IID> 10101 </IID>
    <asignatura_id> CS-101 </asignatura_id>
  </enseña>
  <enseña>
    <IID> 83821 </IID>
    <asignatura_id> CS-101 </asignatura_id>
  </enseña>
  <enseña>
    <IID> 76766 </IID>
    <asignatura_id> BIO-301 </asignatura_id>
  </enseña>
</universidad>
```

Figura 23.1. Representación XML de información de la universidad.

```
<pedido_compra>
  <identificador> P-101 </identificador>
  <comprador>
    <nombre> Cray Z. Coyote </nombre>
    <dirección> Mesa Flats, Route 66, Arizona 12345, USA </dirección>
  </comprador>
  <proveedor>
    <nombre> Acme Supplies </nombre>
    <dirección> 1 Broadway, New York, NY, USA </dirección>
  </proveedor>
  <lista_elementos>
    <elemento>
      <identificador> RS1 </identificador>
      <descripción> Trineo propulsado por cohetes atómicos </descripción>
      <cantidad> 2 </cantidad>
      <precio> 199.95 </precio>
    </elemento>
    <elemento>
      <identificador> SG2 </identificador>
      <descripción> Pegamento fuerte </descripción>
      <cantidad> 1 </cantidad>
      <unidad_de_medida> litro </unidad_de_medida>
      <precio> 29.95 </precio>
    </elemento>
  </lista_elementos>
  <coste_total> 429.85 </coste_total>
  <forma_de_pago> Reembolso </forma_de_pago>
  <forma_de_envío> Avión </forma_de_envío>
</pedido_compra>
```

Figura 23.2. Representación XML de un pedido de compra.

23.2. Estructura de los datos XML

El componente fundamental en un documento XML es el **elemento.** Un elemento es sencillamente un par de etiquetas de inicio y finalización coincidentes y todo el texto que aparece entre ellas.

Los documentos XML deben tener un único elemento **raíz** que abarque al resto de elementos en el documento. En el ejemplo de la Figura 23.1, el elemento <universidad> forma el elemento raíz. Además, los elementos en un documento XML se deben **anidar** adecuadamente. Por ejemplo:

<asignatura> … <nombre_asig> …
</nombre_asig> … </asignatura>

está anidado adecuadamente, mientras que:

<asignatura> … <nombre_asig> …
</asignatura> … </nombre_asig>

no está adecuadamente anidado.

Aunque el anidamiento adecuado es una propiedad intuitiva, es necesario definirla más formalmente. Se dice que el texto aparece **en el contexto de** un elemento si aparece entre la etiqueta de inicio y la etiqueta de finalización de dicho elemento. Las etiquetas están anidadas adecuadamente si toda etiqueta de inicio tiene una única etiqueta de finalización coincidente que está en el contexto del mismo elemento padre.

Observe que el texto puede estar mezclado con los subelementos de otro elemento, como en la Figura 23.3. Como con otras características de XML, esta libertad tiene más sentido en un contexto de procesamiento de documentos que en el contexto de procesamiento de datos y no es particularmente útil para representar en XML datos más estructurados, como es el contenido de las bases de datos.

La capacidad de anidar elementos con otros elementos proporciona una forma alternativa de representar información. La Figura 23.4 muestra una representación de parte de la información de la universidad de la Figura 23.1, pero con los elementos asignatura anidados dentro de los elementos departamento. La representación anidada consigue que sea fácil encontrar todos los cursos que ofrece un departamento. De forma similar, los identificadores de asignaturas que enseña un profesor están anidados dentro de los elementos profesor. Si un profesor enseña más de una asignatura, existirán varios elementos asignatura_id dentro del correspondiente elemento profesor. Los detalles sobre los profesores Brandt y Crick se han omitido en la Figura 23.4 por falta de espacio, pero su estructura es similar a la de Srinivasan.

```
…
  <asignatura>
    Este curso se ofertó por primera vez en 2009.
    <asignatura_id> BIO-399 </asignatura_id>
    <nombre_asig> Biología Computacional </nombre_asig>
    <nombre_dept> Biología </nombre_dept>
    <créditos> 3 </créditos>
  </asignatura>
…
```

Figura 23.3. Mezcla de texto con subelementos.

```
<universidad-1>
  <departamento>
    <nombre_dept> Informática </nombre_dept>
    <edificio> Taylor </edificio>
    <presupuesto> 100000 </presupuesto>
    <asignatura>
      <asignatura_id> CS-101 </asignatura_id>
      <nombre_asig> Intro. a la Informática </nombre_asig>
      <créditos> 4 </créditos>
    </asignatura>
    <asignatura>
      <asignatura_id> CS-347 </asignatura_id>
      <nombre_asig> Conceptos de bases de datos </nombre_asig>
      <créditos> 3 </créditos>
    </asignatura>
  </departamento>
  <departamento>
    <nombre_dept> Biología </nombre_dept>
    <edificio> Watson </edificio>
    <presupuesto> 90000 </presupuesto>
    <asignatura>
      <asignatura_id> BIO-301 </asignatura_id>
      <nombre_asig> Genética </nombre_asig>
      <créditos> 4 </créditos>
    </asignatura>
  </departamento>
  <profesor>
    <IID> 10101 </IID>
    <nombre> Srinivasan </nombre>
    <nombre_dept> Informática </nombre_dept>
    <sueldo> 65000.</sueldo>
    <asignatura_id> CS-101 </asignatura_id>
  </profesor>
</universidad-1>
```

Figura 23.4. Representación XML anidada de información de la universidad.

Aunque la representación anidada es natural en XML, puede generar un almacenamiento de datos redundantes. Por ejemplo, suponga que se almacenan los detalles de las asignaturas que enseña un profesor en el elemento profesor como se muestra en la Figura 23.5. Si el curso lo enseña más de un profesor, la información de dicha asignatura, como el nombre, el departamento y los créditos se almacenarán de forma redundante con cada uno de los profesores asociados a la asignatura.

Las representaciones anidadas se usan ampliamente en las aplicaciones de intercambio de datos XML para evitar las reuniones. Por ejemplo, un pedido de compra almacenaría la dirección completa del emisor y el receptor de una forma redundante en varios pedidos de compra, mientras que una representación normalizada puede requerir una reunión de registros de pedidos de compras con una relación *dirección_empresa* para obtener la información de la dirección.

Además de los elementos, XML especifica la noción de **atributo.** Por ejemplo, el identificador de una asignatura se puede representar como un atributo, como en la Figura 23.6. Los atributos de un elemento aparecen como pares *nombre=valor* antes del cierre «>» de una etiqueta. Los atributos son cadenas de texto y no contienen marcas. Además, los atributos pueden aparecer solamente una vez en una etiqueta dada, al contrario que los subelementos, que pueden estar repetidos.

Observe que en un contexto de construcción de un documento es importante la distinción entre subelemento y atributo (un atributo es implícitamente texto que no aparece en el documento impreso o visualizado). Sin embargo, en las aplicaciones de bases de datos y de intercambio de datos de XML esta distinción es menos relevante y la elección de representar los datos como un atributo o un subelemento es frecuentemente arbitraria. En general, se recomienda utilizar atributos solo para representar identificadores, y almacenar el resto de los datos como subelementos.

Una nota sintáctica final es que un elemento de la forma <elemento></elemento>, que no contiene subelementos o texto, se puede abreviar como <elemento/>; los elementos abreviados pueden, no obstante, contener atributos.

Puesto que los documentos XML se diseñan para su intercambio entre aplicaciones, se tiene que introducir un mecanismo de **espacio de nombres** para permitir a las organizaciones especificar globalmente nombres únicos que se utilicen como marcas de elementos en los documentos. La idea de un espacio de nombres es anteponer cada etiqueta o atributo con un identificador de recursos universal (por ejemplo, una dirección web). Así, por ejemplo, si Yale University deseara asegurar que los documentos XML que crea no duplican las etiquetas usadas por los documentos de otros socios del negocio, se puede anteponer un identificador único con dos puntos a cada nombre de etiqueta. La universidad puede usar una dirección URL como la siguiente:

http://www.yale.edu

como un identificador único. El uso de identificadores únicos largos en cada etiqueta puede ser poco conveniente, por lo que el espacio de nombres estándar proporciona una forma de definir una abreviatura para los identificadores.

En la Figura 23.7 el elemento raíz (universidad) tiene un atributo xmlns:yale, que declara que yale está definido como abreviatura de la URL dada anteriormente. Se puede usar entonces la abreviatura en varias marcas de elementos, como se muestra en la figura.

Un documento puede tener más de un espacio de nombres, declarado como parte del elemento raíz. Se puede entonces asociar elementos diferentes con espacios de nombres distintos. Se puede definir un *espacio de nombres predeterminado* mediante el uso del atributo xmlns, en lugar de xmlns:yale, en el elemento raíz. Los elementos sin un prefijo de espacio de nombres explícito pertenecen entonces al espacio de nombres predeterminado.

Algunas veces es necesario almacenar valores que contienen etiquetas sin que se interpreten como etiquetas XML. Para ello, XML permite esta construcción:

<![CDATA[<asignatura> … </asignatura>]]>

Debido a que el texto <asignatura> está encerrado en CDATA, se trata como datos de texto normal, no como una etiqueta. El término CDATA viene de datos de tipo carácter (*character data*, en inglés).

```
<universidad-2>
  <profesor>
    <ID> 10101 </ID>
    <nombre> Srinivasan </nombre>
    <nombre_dept> Informática</nombre_dept>
    <sueldo> 65000 </sueldo>
    <enseña>
      <asignatura>
        <asignatura_id> CS-101 </asignatura_id>
        <nombre_asig> Intro. a la Informática </nombre_asig>
        <nombre_dept> Informática </nombre_dept>
        <créditos> 4 </créditos>
      </asignatura>
    </enseña>
  </profesor>
  <profesor>
    <ID> 83821 </ID>
    <nombre> Brandt </nombre>
    <nombre_dept> Informática</nombre_dept>
    <sueldo> 92000 </sueldo>
    <enseña>
      <asignatura>
        <asignatura_id> CS-101 </asignatura_id>
        <nombre_asig> Intro. a la Informática </nombre_asig>
        <nombre_dept> Informática </nombre_dept>
        <créditos> 4 </créditos>
      </asignatura>
    </enseña>
  </profesor>
</universidad-2>
```

Figura 23.5. Redundancia en la representación anidada de XML.

```
…
  <asignatura asignatura_id = «CS-101»>
    <nombre_asig> Intro. a la Informática </nombre_asig>
    <nombre_dept> Informática </nombre_dept>
    <créditos> 4 </créditos>
  </asignatura>
…
```

Figura 23.6. Uso de atributos.

```
<universidad xmlns:yale =«http://www.yale.edu»>
  …
  <yale:asignatura>
    <yale:asignatura_id> CS-101 </yale:asignatura_id>
    <yale:nombre_asig> Intro. a la Informática </yale:nombre_asig>
    <yale:nombre_dept> Informática </yale:nombre_dept>
    <yale:créditos> 4 </yale:créditos>
  </yale:asignatura>
  …
</universidad>
```

Figura 23.7. Nombres únicos de etiqueta mediante el uso de espacios de nombres.

23.3. Esquema de documentos XML

Las bases de datos contienen esquemas que se usan para restringir qué información se puede almacenar en la base de datos y los tipos de datos de la información almacenada. En cambio, los documentos XML se pueden crear de forma predeterminada sin un esquema asociado. Un elemento puede tener entonces cualquier subelemento o atributo. Aunque dicha libertad puede ser aceptable, algunas veces, dada la naturaleza autodescriptiva del formato de datos, no es útil; generalmente cuando los documentos XML se deben procesar automáticamente como parte de una aplicación o incluso cuando se va a dar formato en XML a grandes cantidades de datos relacionados.

Aquí se describirá el primer lenguaje de definición de esquemas incluido como parte de la norma XML, la definición de tipos de documentos (*Document Type Definition*), así como su sustituto *XML Schema*, definido más recientemente. También se usa otro lenguaje de definición de esquemas denominado Relax NG, pero no se estudia aquí; para obtener más información sobre Relax NG consulte las referencias en el apartado de notas bibliográficas.

23.3.1. Definición de tipos de documentos

La **definición de tipos de documentos** (*document type definition*: DTD) es una parte opcional de un documento XML. El propósito principal de DTD es similar al de un esquema: restringir el tipo de información presente en el documento. Sin embargo, DTD no restringe en realidad los tipos en el sentido de tipos básicos como entero o cadena. En su lugar, solamente restringe el aspecto de subelementos y atributos en un elemento. DTD es principalmente una lista de reglas que indican el patrón de subelementos que aparecen en un elemento. La Figura 23.8 muestra una parte de una DTD de ejemplo de un documento de información universitaria; el documento XML de la Figura 23.1 es conforme a esa DTD.

Cada declaración está en la forma de una expresión regular para los subelementos de un elemento. Así, en la DTD de la Figura 23.8 un elemento de la universidad consta de uno o más elementos asignatura, departamento o profesor; el operador | especifica «o», mientras que el operador + especifica «uno o más». Aunque no se muestra aquí, el operador ∗ se usa para especificar «cero o más», mientras que el operador ? se usa para especificar un elemento opcional (es decir, «cero o uno»).

El elemento asignatura se define para contener los subelementos asignatura_id, nombre_asig, nombre_dept y créditos (en ese orden). De forma similar, departamento y profesor tienen los atributos en su esquema relacional definidos como subelementos en la DTD.

Finalmente, se declara a los elementos asignatura_id, nombre_asig, nombre_dept, créditos, edificio, presupuesto, IID, nombre y sueldo del tipo #PCDATA. La palabra clave #PCDATA indica dato de texto; su nombre deriva históricamente de «parsed character data» (datos analizados de tipo carácter). Otros dos tipos especiales de declaraciones son empty (vacío), que indica que el elemento no tiene ningún contenido, y any (cualquiera), que indica que no hay restricción sobre los subelementos del elemento; es decir, cualquier elemento, incluso los no mencionados en la DTD, puede ser subelemento del elemento. La ausencia de una declaración para un subelemento es equivalente a declarar explícitamente el tipo como any.

Los atributos permitidos para cada elemento también se declaran en la DTD. Al contrario que en los subelementos no se impone ningún orden a los atributos. Los atributos se pueden especificar del tipo CDATA, ID, IDREF o IDREFS; el tipo CDATA simplemente dice que el atributo contiene datos de caracteres, mientras que los otros tres no son tan sencillos; se explicarán detalladamente en breve. Por ejemplo, la siguiente línea de una DTD especifica que el elemento asignatura tiene un atributo del tipo asignatura_id y que tiene que tener un valor en este atributo:

```
<!ATTLIST asignatura asignatura_id CDATA #REQUIRED>
```

Los atributos deben tener una declaración de tipo y una declaración predeterminada. La declaración predeterminada puede consistir en un valor predeterminado para el atributo o #REQUIRED, lo que quiere decir que se debe especificar un valor para el atributo en cada elemento, #IMPLIED, lo que significa que no se ha proporcionado ningún valor predeterminado y se puede omitir este atributo en el documento. Si un atributo tiene un valor predeterminado, para cada elemento que no tenga especificado un valor para el atributo el valor se rellena automáticamente cuando se lee el documento XML.

Un atributo de tipo ID proporciona un identificador único para el elemento; un valor de un atributo ID de un elemento no debe aparecer en ningún otro elemento del mismo documento. Solo se permite que un único atributo de un elemento sea de tipo ID. (Se ha renombrado el atributo *ID* de la relación *profesor* a IID en la representación XML, para evitar la confusión con el tipo ID).

Un atributo del tipo IDREF es una referencia a un elemento; el atributo debe contener un valor que aparezca en el atributo ID de algún elemento en el documento. El tipo IDREFS permite una lista de referencias, separadas por espacios.

La Figura 23.9 muestra una DTD de ejemplo en la que los identificadores de asignatura, departamento y profesor se representan mediante los atributos ID e IDREFS. Los elementos asignatura usan asignatura_id como atributo identificador; para ello, asignatura_id se ha convertido en un atributo de asignatura, en lugar de un subelemento. Además, cada elemento asignatura contiene un IDREFS del departamento correspondiente a la asignatura, y un atributo IDREFS profesores que identifica a los profesores que enseñan la asignatura. El elemento departamento tiene un atributo identificador llamado nombre_dept. Los elementos profesor tienen un atributo identificador llamado IID y un atributo IDREF nombre_dept que identifica al departamento al que pertenece el profesor.

La Figura 23.10 muestra un ejemplo de documento XML basado en la DTD de la Figura 23.9.

```
<!DOCTYPE universidad [
  <!ELEMENT universidad ((departamento|asignatura|
                          profesor|enseña)+)>
  <!ELEMENT departamento (nombre_dept, edificio, presupuesto)>
  <!ELEMENT asignatura (asignatura_id, nombre_asig,
                        nombre_dept, créditos)>
  <!ELEMENT profesor (IID, nombre, nombre_dept, sueldo)>
  <!ELEMENT enseña (IID, asignatura_id)>
  <!ELEMENT nombre_dept(#PCDATA)>
  <!ELEMENT edificio(#PCDATA)>
  <!ELEMENT presupuesto(#PCDATA)>
  <!ELEMENT asignatura_id (#PCDATA)>
  <!ELEMENT nombre_asig (#PCDATA)>
  <!ELEMENT créditos(#PCDATA)>
  <!ELEMENT IID(#PCDATA)>
  <!ELEMENT nombre(#PCDATA)>
  <!ELEMENT sueldo(#PCDATA)>
] >
```

Figura 23.8. Ejemplo de una DTD.

Los atributos ID e IDREF desempeñan la misma función que los mecanismos de referencia en las bases de datos orientadas a objetos y las bases de datos relacionales orientadas a objetos, permitiendo la construcción de relaciones de datos complejas.

```
<!DOCTYPE universidad-3 [
  <!ELEMENT universidad ((departamento|asignatura|profesor)+)>
  <!ELEMENT departamento (edificio, presupuesto)>
  <!ATTLIST departamento
    nombre_dept ID #REQUIRED >
  <!ELEMENT asignatura (nombre_asig, créditos)>
  <!ATTLIST asignatura
    asignatura_id ID #REQUIRED
    nombre_dept IDREF #REQUIRED
    profesores IDREFS #IMPLIED>
  <!ELEMENT profesor (nombre, sueldo)>
  <!ATTLIST profesor
    IID ID #REQUIRED>
    nombre_dept IDREF #REQUIRED>
  ...declaraciones para nombre_asig, créditos, edificio, presupuesto,
    nombre y sueldo ...
] >
```

Figura 23.9. DTD con tipos de atributo ID e IDREFS.

```
<universidad-3>
  <departamento nombre_dept=«Informática»>
    <edificio> Taylor </edificio>
    <presupuesto> 100000 </presupuesto>
  </departamento>
  <departamento nombre_dept=«Biología»>
    <edificio> Watson </edificio>
    <presupuesto> 90000 </presupuesto>
  </departamento>
  <asignatura asignatura_id=«CS-101» nombre_dept=«Informática»
              profesores=«10101 83821»>
    <nombre_asig> Intro. a la Informática</nombre_asig>
    <créditos> 4 </créditos>
  </asignatura>
  <asignatura asignatura_id=«BIO-301» nombre_dept=«Biología»
              profesores =«76766»>
    <nombre_asig> Genética </nombre_asig>
    <créditos> 4 </créditos>
  </asignatura>
  <profesor IID=«10101» nombre_dept=«Informática»>
    <nombre> Srinivasan </nombre>
    <sueldo> 65000 </sueldo>
  </profesor>
  <profesor IID=«83821» nombre_dept=«Informática»>
    <nombre> Brandt </nombre>
    <sueldo> 72000 </sueldo>
  </profesor>
  <profesor IID=«76766» nombre_dept=«Biología»>
    <nombre> Crick </nombre>
    <sueldo> 72000 </sueldo>
  </profesor>
</universidad-3>
```

Figura 23.10. Datos XML con los atributos ID e IDREF.

Las definiciones de tipos de documentos están fuertemente relacionadas con la herencia del formato del documento XML; por ello, no son adecuadas para servir como estructura de tipos de XML para aplicaciones de procesamiento de datos. No obstante, se están definiendo un gran número de formatos de intercambio de datos en términos de DTD, puesto que fueron parte original de la norma. Veamos algunas limitaciones de las DTD como mecanismo de esquema:

- No se puede declarar el tipo de cada elemento y de cada atributo de texto. Por ejemplo, el elemento sueldo no se puede restringir para que sea un número positivo. La falta de tal restricción es problemática para las aplicaciones de procesamiento e intercambio de datos, ya que deben contener código para verificar los tipos de los elementos y atributos.
- Es difícil usar el mecanismo DTD para especificar conjuntos desordenados de subelementos. El orden es rara vez importante para el intercambio de datos (al contrario que en el diseño de documentos, donde es crucial). Aunque la combinación de la alternativa (la operación |) y las operaciones $*$ y $+$, como en la Figura 23.8, permite la especificación de colecciones desordenadas de marcas, es mucho más complicado especificar que cada marca pueda aparecer solamente una vez.
- Hay una falta de tipos en ID e IDREFS. Por ello, no hay forma de especificar el tipo de elemento al cual se debería referir un atributo IDREF o IDREFS. En consecuencia, la DTD de la Figura 23.9 no evita que el atributo «nombre_dept» de un elemento asignatura se refiera a otras asignaturas, aunque esto no tenga sentido.

23.3.2. XML Schema

Un intento de reparar las deficiencias del mecanismo DTD produjo el desarrollo de un lenguaje de esquema más sofisticado, **XML Schema.** Se presenta aquí una descripción general de XML Schema y se mencionan algunas áreas en las cuales mejora a las DTD.

XML Schema define varios tipos predefinidos como string, integer, decimal, date y boolean. Además, permite tipos definidos por el usuario, que pueden ser tipos más simples con restricciones añadidas, o tipos complejos construidos con constructores como complexType y sequence.

La Figura 23.11 muestra cómo la DTD de la Figura 23.8 se puede representar mediante XML Schema. A continuación se describen las características de XML Schema que aparecen en esa figura.

Lo primero que es preciso resaltar es que las propias definiciones en XML Schema se especifican en sintaxis XML, usando varias etiquetas definidas en XML Schema. Para evitar conflictos con las etiquetas definidas por los usuarios, se antepone a la etiqueta XML Schema la etiqueta de espacio de nombres «xs:»; este prefijo se asocia con el espacio de nombres de XML Schema mediante la especificación xmlns:xs en el elemento raíz:

```
<xs:schema xmlns:xs=«http://www.w3.org/2001/XMLSchema»>
```

Observe que se podría usar cualquier prefijo de espacio de nombres en lugar de xs; por tanto, se podrían reemplazar todas las apariciones de «xs:» por «xsd:» sin cambiar el significado de la definición del esquema. A todos los tipos definidos en XML Schema se les debe anteponer este prefijo de espacio de nombres.

El primer elemento es el elemento raíz universidad, cuyo tipo es TipoUniversidad, que se declara posteriormente. En el ejemplo se definen después los tipos de los elementos departamento, asignatura, profesor y enseña. Observe que cada uno de ellos se especifica con un elemento con la etiqueta xs:element, cuyo cuerpo contiene la definición del tipo.

El tipo de departamento se define como complejo, se especifica seguidamente y se define como una secuencia de elementos nombre_dept, edificio y presupuesto. Cualquier tipo que tenga atributos o subelementos anidados se tiene que especificar como tipo complejo.

Alternativamente se puede especificar que el tipo de un elemento sea un tipo predefinido con el atributo type; observe el uso de los tipos de XML Schema xs:string y xs:decimal para restringir los tipos de los elementos de datos tales como nombre_dept y créditos.

Finalmente, el ejemplo define el tipo TipoUniversidad para contener cero o más apariciones de departamento, asignatura, profesor y enseña. Observe el uso de ref para especificar la aparición de un elemento definido anteriormente. XML Schema puede definir el número mínimo y máximo de apariciones de un subelemento mediante minOccurs y maxOccurs. El valor predeterminado para las apariciones máxima y mínima es 1, por lo que se tiene que especificar explícitamente para permitir cero o más departamento, asignatura, profesor y enseña.

Los atributos se especifican usando la etiqueta xs:attribute. Por ejemplo, se podría haber definido nombre_dept como atributo añadiendo:

```
<xs:attribute nombre = «nombre_dept»/>
```

dentro de la declaración del elemento departamento. Al añadir el atributo use = «required» a la especificación anterior se declara que se debe especificar el atributo, mientras que el valor predeterminado de use es optional. Las especificaciones de atributos pueden aparecer directamente dentro de la especificación complexType, aunque los elementos se aniden dentro de una especificación de secuencia.

Se puede usar el elemento xs:complexType para crear tipos complejos con nombre; la sintaxis es la misma que la utilizada en el elemento xs:complexType de la Figura 23.11, salvo que se añade un atributo name = nombreTipo al elemento xs:complexType, donde nombreTipo es el nombre que se desea para el tipo. Se puede usar entonces el nombre del tipo para especificar el tipo de un elemento usando el atributo type, al igual que se han usado xs:decimal y xs:string en el ejemplo.

Además de definir tipos, un esquema relacional también permite la especificación de restricciones. XML Schema permite la especificación de claves y referencias a claves, que se corresponden con las definiciones de clave primaria y clave externa de SQL. En SQL, una restricción de clave primaria o de unicidad asegura que los valores del atributo no se dupliquen en la relación. En el contexto de XML es necesario definir un ámbito en el que los valores sean únicos y formen una clave. La declaración selector es una expresión de ruta que define el ámbito de la restricción, y las declaraciones field especifican los elementos o atributos que forman la clave.[1] Para especificar que nombre_dept forma una clave de los elementos departamento en el elemento raíz universidad se añade la siguiente especificación de restricción a la definición del esquema:

```
<xs:key name = «claveDept»>
  <xs:selector xpath = «/universidad/departamento»/>
  <xs:field xpath = «nombre_dept»/>
</xs:key>
```

Se puede definir también la restricción de clave externa desde asignatura a departamento como sigue:

```
<xs:name = «claveExtAsignaturaDept» refer=«claveDept»>
  <xs:selector xpath = «/universidad/asignatura»/>
  <xs:field xpath = «nombre_dept»/>
</xs:keyref>
```

Observe que el atributo refer especifica el nombre de la declaración de clave a la que se hace referencia, mientras que la especificación field identifica los atributos que hacen la referencia.

1 En el ejemplo solo se usan expresiones path con una sintaxis familiar. XML tiene una rica sintaxis para las expresiones path, denominada XPath, que se tratará en la Sección 23.4.2.

XML Schema ofrece varias ventajas con respecto a las DTD, y actualmente se usa mucho. Entre las ventajas que se han visto en los ejemplos anteriores se encuentran las siguientes:

- Permite que el texto que aparece en los elementos esté restringido a tipos específicos, tales como tipos numéricos en formatos específicos o tipos más complejos como secuencias de elementos de otros tipos.
- Permite crear tipos definidos por el usuario.
- Permite restricciones de unicidad y de clave externa.
- Está integrado con espacios de nombres para permitir a diferentes partes de un documento adaptarse a esquemas diversos.

```
<xs:schema xmlns:xs=«http://www.w3.org/2001/XMLSchema»>
<xs:element name=«universidad» type=«TipoUniversidad» />
<xs:element name=«departamento»>
  <xs:complexType>
    <xs:sequence>
      <xs:element name=«nombre_dept» type=«xs:string»/>
      <xs:element name=«edificio» type=«xs:string»/>
      <xs:element name=«presupuesto» type=«xs:decimal»/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name=«asignatura»>
  <xs:element name=«asignatura_id» type=«xs:string»/>
  <xs:element name=«nombre_asig» type=«xs:string»/>
  <xs:element name=«nombre_dept» type=«xs:string»/>
  <xs:element name=«créditos» type=«xs:decimal»/>
</xs:element>
<xs:element name=«profesor»>
  <xs:complexType>
    <xs:sequence>
      <xs:element name=«IID» type=«xs:string»/>
      <xs:element name=«nombre» type=«xs:string»/>
      <xs:element name=«nombre_dept» type=«xs:string»/>
      <xs:element name=«sueldo» type=«xs:decimal»/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name=«enseña»>
  <xs:complexType>
    <xs:sequence>
      <xs:element name=«IID» type=«xs:string»/>
      <xs:element name=«asignatura_id» type=«xs:string»/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name=«TipoUniversidad»>
  <xs:sequence>
    <xs:element ref=«departamento» minOccurs=«0» maxOccurs=
                «unbounded»/>
    <xs:element ref=«asignatura» minOccurs=«0» maxOccurs=
                «unbounded»/>
    <xs:element ref=«profesor» minOccurs=«0» maxOccurs=
                «unbounded»/>
    <xs:element ref=«enseña» minOccurs=«0» maxOccurs=
                «unbounded»/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Figura 23.11. Versión XML Schema de la DTD de la Figura 23.8.

Además de las características estudiadas, XML Schema soporta otras características que las DTD no soportan, como:

- Permite restringir tipos para crear tipos especializados; por ejemplo, especificando valores máximos y mínimos.
- Permite extender los tipos complejos mediante el uso de una forma de herencia.

Esta descripción de XML Schema es solo una descripción general; para más información, consulte las notas bibliográficas.

```
<universidad-1>
{
   for $d in /universidad/departamento
   return
      <departamento>
         {$d/*}
         {for $c in /universidad/asignatura[nombre_dept = $d/nombre_dept]
                  return $c}
      </departamento>
}
{
   for $i in /universidad/profesor
   return
      <profesor>
         {$i/*}
         {for $c in /universidad/enseña[IID = $i/IID]
                  return $c/asignatura_id }
      </profesor>
}
</universidad-1>
```

Figura 23.12. Creación de estructuras anidadas en XQuery.

23.4. Consulta y transformación

Dado el creciente número de aplicaciones que usan XML para intercambiar, transmitir y almacenar datos, cada vez están siendo más importantes las herramientas para una gestión efectiva de datos XML. En particular, las herramientas para consultar y transformar los datos XML son esenciales para extraer información de grandes cuerpos de datos XML y para convertir los datos entre distintas representaciones (esquemas) en XML. Al igual que la salida de una consulta relacional es una relación, la salida de una consulta XML puede ser un documento XML. En consecuencia, la consulta y la transformación se pueden combinar en una única herramienta.

En esta sección se describen los lenguajes XPath y XQuery:

- XPath es un lenguaje para expresiones de rutas y es realmente un bloque constructivo para XQuery.
- XQuery es el lenguaje normalizado para la consulta de datos XML. Se ha modelado después de SQL, pero es significativamente diferente, ya que tiene que manejar datos XML anidados. También incorpora expresiones XPath.

El lenguaje XSLT es otro lenguaje diseñado para transformar XML. Sin embargo, se utiliza principalmente para dar formato a documentos, en lugar de para aplicaciones de gestión de datos, por lo que no se tratará en este libro.

En la sección de herramientas del final del capítulo se proporcionan referencias a software que se puede utilizar para ejecutar consultas escritas en XPath y XQuery.

23.4.1. Modelo de árbol de XML

En todos estos lenguajes se utiliza un **modelo de árbol** de datos XML. Cada documento XML se modela como un **árbol,** con **nodos** correspondientes a los elementos y a los atributos. Los nodos de los elementos pueden tener nodos hijos, que pueden ser subelementos o atributos del elemento. De igual forma, cada nodo (ya sea de atributo o de elemento) distinto del elemento raíz tiene un nodo padre, que es un elemento. El orden de los elementos y de los atributos en el documento XML se modela ordenando los nodos hijos del árbol. Los términos padre, hijo, ascendiente, descendiente y hermano se utilizan en el modelo de árbol de datos XML.

El contenido textual de un elemento se puede modelar como un nodo de texto hijo del elemento. Los elementos que contienen texto descompuesto por subelementos intervinientes pueden tener varios nodos de texto hijo. Por ejemplo, un elemento que contenga «Este es un <bold> gran </bold> libro» contendría un subelemento hijo correspondiente al elemento bold y dos nodos de texto hijos correspondientes a «Este es un» y a «libro». Puesto que dichas estructuras no se usan habitualmente en los datos de las bases de datos, se asumirá que los elementos no contienen a la vez texto y subelementos.

23.4.2. XPath

XPath trata partes de los documentos XML mediante expresiones de rutas. Se puede ver el lenguaje como una extensión a las expresiones de rutas sencillas en las bases de datos orientadas a objetos y relacionales orientadas a objetos (véase la Sección 22.6). La versión actual de XPath es la 2.0 y nuestra descripción se basa en ella.

Cada **expresión de ruta** de XPath es una secuencia de pasos de ubicación separados por «/» (en lugar de por el operador «.», que separa los pasos en SQL). El resultado de una expresión de ruta es un conjunto de nodos. Por ejemplo, en el documento de la Figura 23.10, la expresión XPath:

/universidad-3/profesor/nombre

devuelve estos elementos:

<nombre>Srinivasan</nombre>
<nombre>Brandt</nombre>

La expresión:

/universidad-3/profesor/nombre/text()

devuelve los mismos nombres, pero sin las etiquetas que los rodean.

Las expresiones de ruta se evalúan de izquierda a derecha. Al igual que en una jerarquía de directorios, la barra ´/´ inicial indica la raíz del documento (observe que es una raíz abstracta «por encima de» <universidad-3>, que es la etiqueta de documento).

Al evaluar una expresión de ruta, el resultado de la ruta en cualquier punto consiste en un conjunto ordenado de nodos del documento. Inicialmente, el conjunto de elementos «real» contiene un solo nodo, la raíz abstracta. Cuando el siguiente paso de una expresión de ruta es un nombre de elemento, como profesor, el resultado del paso consiste en los nodos correspondientes a elementos del nombre especificado que son los hijos de los elementos en el conjunto actual de elementos. Estos nodos serán el conjunto actual de elementos para el siguiente paso de la evaluación de la expresión de ruta. Por tanto, la expresión:

/universidad-3

devuelve un único nodo correspondiente a la etiqueta:

<universidad-3>

mientras que:

/universidad-3/profesor

devuelve los dos nodos correspondientes a los elementos:

profesor

que son hijos del nodo:

universidad-3

El resultado de una expresión de ruta es, entonces, el conjunto de nodos tras el último paso de la evaluación de la expresión de ruta. Los nodos que se devuelven en cada paso aparecen en el mismo orden en que aparecen en el documento.

Ya que varios hijos pueden tener el mismo nombre, el número de nodos en el conjunto puede aumentar o disminuir en cada paso. También se puede acceder a los valores de los atributos usando el símbolo «@». Por ejemplo, /universidad-3/asignatura/@asignatura_id devuelve un conjunto con todos los valores de los atributos asignatura_id de los elementos asignatura. De forma predeterminada no se siguen los vínculos IDREF; posteriormente veremos cómo tratar con IDREF.

XPath soporta otras características:

- Los predicados de selección pueden seguir a cualquier paso en una ruta y se escriben entre corchetes. Por ejemplo:

  /universidad-3/asignatura[créditos >= 4]

  devuelve los elementos asignatura con un valor de créditos mayor o igual a 4, mientras que:

  /universidad-3/asignatura[créditos >= 4]/@asignatura_id

  devuelve los identificadores de asignatura de dichas asignaturas.

  Se puede comprobar la existencia de un subelemento listándolo sin ningún operador de comparación; de hecho, si se quita simplemente «>= 4» de la expresión anterior, devolvería los identificadores de asignatura de todas las asignaturas que tengan un subelemento créditos, independientemente de su valor.
- XPath proporciona varias funciones que se pueden usar como parte de predicados, incluidas la comprobación de la posición del nodo actual en el orden de los hermanos y la función de agregación count(), que cuenta el número de nodos coincidentes con la expresión a la que se aplica. Por ejemplo, sobre la representación de XML de la Figura 23.5, la expresión de ruta:

  /universidad-2/profesor[count(./enseña/asignatura)> 2]

  devuelve los profesores que enseñan más de dos asignaturas. Se pueden usar las conectivas lógicas and y or en los predicados y la función not(…) para la negación.
- La función id(«fu») devuelve el nodo (si existe) con un atributo del tipo ID cuyo valor sea «fu». La función id se puede incluso aplicar a conjuntos de referencias o a cadenas que contengan referencias múltiples separadas por espacios, tales como IDREFS. Por ejemplo, la ruta:

  /universidad-3/asignatura/id(@nombre_dept)

  devuelve todos los departamentos referenciados con el atributo nombre_dept de los elementos asignatura, mientras que:

  /universidad-3/asignatura/id(@profesores)

  devuelve el elemento profesor al que se refiere en el atributo profesores de los elementos asignatura.
- El operador | permite unir resultados de expresiones. Por ejemplo, si los datos usan el esquema de la Figura 23.10, se podría definir la unión de las asignaturas Informática y Biología utilizando la expresión:

  /universidad-3/asignatura[@nombre_dept=«Informática»] |
  /universidad-3/asignatura[@nombre_dept=«Biología»]

  Sin embargo, el operador | no se puede anidar dentro de otros operadores. También merece la pena destacar que los nodos de la unión se devuelven en el orden en el que aparecen en el documento.
- Una expresión XPath puede saltar varios niveles de nodos mediante el uso de «//». Por ejemplo, la expresión /universidad-3//nombre encuentra todos los elementos nombre en *cualquier lugar* por debajo del elemento /universidad-3, independientemente del elemento en el que esté contenido y del número de niveles de inclusión presentes entre los elementos universidad-3 y nombre. Este ejemplo muestra la posibilidad de buscar los datos necesarios sin un conocimiento completo del esquema.
- Un paso en la ruta no selecciona solamente los hijos de los nodos del conjunto actual de nodos. En realidad, esto es solamente una de las distintas direcciones que puede tomar un paso en la ruta, tales como padres, hermanos, ascendientes y descendientes. Se omiten los detalles, pero observe que «//», descrito anteriormente, es una forma abreviada de especificar «todos los descendientes», mientras que «..» especifica el padre.
- La función predeterminada doc(nombre) devuelve la raíz de un documento con nombre; el nombre puede ser un nombre de archivo o una URL. La raíz que devuelve la función se puede usar en una expresión de ruta para acceder a los contenidos del documento. Por tanto, se puede aplicar una expresión de ruta en un documento especificado, en lugar de aplicarse al documento actual predeterminado.

  Por ejemplo, si los datos universitarios de nuestro ejemplo estuviesen en el archivo «universidad.xml», la siguiente expresión de ruta devolvería todos los departamentos de la universidad:

  doc(«universidad.xml»)/universidad/departamento

  La función collection(nombre) es similar a doc, pero devuelve una colección de documentos identificada por nombre. Se puede utilizar la función collection, por ejemplo, para abrir una base de datos de XML, que se puede ver como una colección de documentos; el siguiente paso de la expresión XPath seleccionaría el documento o documentos de la colección.

  En la mayoría de los ejemplos, supondremos que las expresiones se evalúan en el contexto de una base de datos, que implícitamente proporcionan una colección de «documentos» sobre los que se evalúan las expresiones XPath. En estos casos, no se necesita utilizar las funciones doc ni collection.

23.4.3. XQuery

El consorcio W3C (World Wide Web Consortium) ha desarrollado XQuery como lenguaje de consulta normalizado de XML. La descripción se basa en XQuery 1.0, liberada por W3C el 23 de enero de 2007.

23.4.3.1. Expresiones FLWOR

Las consultas XQuery se modelaron después de SQL, pero difieren significativamente de él. Se organizan en cinco secciones: **for, let, where, order by** y **return.** Se conocen como expresiones «FLWOR» (pronunciado «flauer», como «flor» en inglés), cuyas letras indican las cinco secciones.

A continuación se muestra una expresión FLWOR sencilla que devuelve los identificadores de asignatura de las asignaturas con más de tres créditos, basada en el documento XML de la Figura 23.10, que usa ID e IDREFS:

```
for $x in /universidad-3/asignatura
let $asignaturaId := $x/@asignatura_id
where $x/créditos > 3
return <asignatura_id> {$ asignaturaId} </asignatura_id>
```

La cláusula **for** es como la cláusula **from** de SQL y especifica variables cuyos valores son los resultados de expresiones XPath. Cuando se especifica más de una variable, los resultados incluyen el producto cartesiano de los valores posibles que las variables pueden tomar, al igual que la cláusula **from** de SQL.

La cláusula **let** simplemente permite que se asigne el resultado de expresiones XPath al nombre de las variables por simplicidad de representación. La cláusula **where,** como la cláusula **where** de SQL, ejecuta comprobaciones adicionales sobre las tuplas reunidas de la cláusula **for.** La cláusula **order by,** al igual que la cláusula **order by** de SQL, permite la ordenación de la salida. Finalmente, la cláusula **return** permite la construcción de resultados en XML.

Una consulta FLWOR no necesita tener todas las cláusulas; por ejemplo, una consulta puede contener simplemente las cláusulas **for** y **return** y omitir **let, where** y **order by.** La consulta XQuery anterior no contiene ninguna cláusula **order by.** De hecho, dado que esta consulta es sencilla, se puede prescindir fácilmente de la cláusula **let,** y la variable $asignaturaID de la cláusula **return** se podría remplazar por $x/@asignatura_id. Observe además que, puesto que la cláusula **for** usa expresiones XPath, se pueden producir selecciones en la expresión XPath. Por ello, se puede conseguir una consulta equivalente solo con cláusulas **for** y **return:**

```
for $x in /universidad-3/asignatura[créditos > 3]
return <asignatura_id> {$x/@asignatura_id} </asignatura_id>
```

Sin embargo, la cláusula **let** ayuda a simplificar consultas complejas. Observe también que las variables asignadas por la cláusula **let** pueden contener secuencias con varios elementos o valores, si la expresión de ruta de la parte derecha los devuelve.

Observe el uso de las llaves («{}») en la cláusula **return.** Cuando XQuery encuentra un elemento como <asignatura_id> al comienzo de una expresión, trata su contenido como texto XML normal, excepto las partes encerradas entre llaves, que se evalúan como expresiones. Así, si se hubiesen omitido las llaves en la anterior cláusula **return,** el resultado contendría varias copias de la cadena «$x/@asignatura_id», cada una de ellas encerrada en una etiqueta asignatura_id. Los contenidos dentro de las llaves se tratan, sin embargo, como expresiones a evaluar. Este convenio se aplica incluso con las llaves entre comillas. Por tanto, se podría modificar la consulta anterior para devolver un elemento con una etiqueta asignatura, con el identificador de la asignatura como atributo, reemplazando la cláusula **return** por:

```
return <asignatura asignatura_id=«{$x/@asignatura_id}» />
```

XQuery proporciona otra forma de construir elementos usando los constructores **element** y **attribute.** Por ejemplo, si la cláusula **return** de la consulta anterior se reemplaza por la siguiente cláusula **return,** la consulta devolvería elementos asignatura y nombre_dept como atributos y nombre_asig y créditos como subelemento.

```
return element asignatura {
   attribute asignatura_id {$x/@asignatura_id},
   attribute nombre_dept {$x/nombre_dept},
   element nombre_asig {$x/nombre_asig},
   element créditos {$x/créditos}
}
```

Observe que, al igual que sucedía antes, las llaves son necesarias para tratar una cadena como una expresión a evaluar.

23.4.3.2. Reuniones

Las reuniones se especifican en XQuery de forma parecida a SQL. La reunión de los elementos asignatura, profesor y enseña de la Figura 23.1 se puede escribir en XQuery de la siguiente forma:

```
for $c in /universidad/asignatura,
   $i in /universidad/profesor,
   $t in /universidad/enseña
where $c/asignatura_id= $t/asignatura_id
   and $t/IID = $i/IID
return <asignatura profesor> {$c $i} </asignatura profesor>
```

La misma consulta se puede expresar con las selecciones especificadas como selecciones XPath:

```
for $c in /universidad/asignatura,
   $i in /universidad/profesor,
   $t in /universidad/enseña[$c/asignatura_id= $t/asignatura_id
      and $t/IID = $i/IID]
return <asignatura profesor> {$c $i} </asignatura profesor>
```

Las expresiones de ruta en XQuery son las mismas expresiones de ruta en XPath2.0. Las expresiones de ruta pueden devolver un único valor o elemento, o una secuencia de ellos. Sin información de esquema no se puede inferir si una expresión de ruta devuelve un único valor o una secuencia de valores. Tales expresiones pueden participar en operaciones de comparación tales como =, < y >=.

XQuery tiene una definición interesante de las operaciones de comparación sobre secuencias. Por ejemplo, la expresión $x/créditos > 3 tendría la interpretación usual si el resultado de $x/créditos fuese un único valor, pero si el resultado es una secuencia que contiene varios valores, la expresión se evalúa a cierto si al menos uno de los valores es mayor que 3. Análogamente, la expresión $x/créditos = $y/créditos es cierta si alguno de los valores devueltos por la primera expresión es igual a cualquier otro valor de la segunda. Si este comportamiento no es apropiado, se pueden usar las operaciones eq, ne, lt, gt, le, ge en su lugar, que generan un error si cualquiera de sus entradas es una secuencia de varios valores.

23.4.3.3. Consultas anidadas

Las expresiones FLWOR de XQuery se pueden anidar en la cláusula **return** con el fin de generar anidamientos de elementos que no aparecen en el documento origen. Por ejemplo, la estructura XML mostrada en la Figura 23.4, con los elementos de asignatura anidados en elementos departamento, se puede generar a partir de la estructura de la Figura 23.1 mediante la consulta que se muestra en la Figura 23.12.

La consulta también introduce la sintaxis $c/*, que se refiere a todos los hijos del nodo (o secuencia de nodos), ligados a la variable $c. De forma similar, $c/text() devuelve el contenido textual de un elemento, sin las etiquetas.

XQuery proporciona funciones de agregación como sum() y count() que se pueden aplicar a secuencias de elementos o valores. La función distinct-values() aplicada sobre una secuencia devuelve una secuencia sin duplicados. La secuencia (colección) de valores devueltos por una expresión de ruta puede tener algunos valores repetidos porque se repiten en el documento, aunque un resultado de una expresión XPath pueda contener, a lo sumo, una aparición de cada nodo en el documento. XQuery soporta muchas otras funciones; consulte las referencias en las notas bibliográficas para más información. En realidad estas funciones son comunes a XPath 2.0 y XQuery, y se pueden utilizar en cualquier expresión de ruta de XPath.

Para evitar conflictos de espacio de nombre, las funciones se asocian con un espacio de nombres:

http://www.w3.org/2005/xpath-functions

que tiene el prefijo predeterminado fn para el espacio de nombres. Por tanto, estas funciones se pueden reconocer sin ambigüedades como fn:sum o fn:count.

Aunque XQuery no proporciona un constructor **group by,** las consultas de agregación se pueden escribir usando funciones de agregado sobre expresiones de ruta o FLWOR anidadas dentro de la cláusula **return.**

Por ejemplo, la siguiente consulta sobre el esquema XML de universidad calcula el sueldo total de todos los profesores de todos los departamentos.

```
for $d in /universidad/departamento
return
  <departamento-total-sueldo>
    <nombre_dept> {$d/nombre_dept} </nombre_dept>
    <total_sueldo> {fn:sum(
      for $i in /universidad/profesor[nombre_dept =
                $d/nombre_dept]
      return $i/sueldo
    ) } </total_sueldo>
  </departamento-total-sueldo>
```

23.4.3.4. Ordenación de resultados

En XQuery los resultados se pueden ordenar con la cláusula **order by.** Por ejemplo, esta consulta tiene como salida todos los elementos profesor ordenados por el subelemento nombre:

```
for $i in /universidad/profesor
order by $i/nombre
return <profesor> {$i/*} </profesor>
```

Para ordenar de forma decreciente se utiliza **order by** $i/nombre **descending.**

La ordenación se puede realizar en varios niveles de anidamiento. Por ejemplo, se puede obtener una representación anidada de la información de la universidad ordenada según nombre de los departamentos del cliente, con las asignaturas ordenadas por el identificador de la asignatura, de la siguiente forma:

```
<universidad-1> {
  for $d in /universidad/departamento
  order by $d/nombre_dept
  return
    <departamento>
      {$d/*}
      {for $c in /universidad/asignatura[nombre_dept =
                $d/nombre_dept]
      order by $c/asignatura_id
      return <asignatura> {$c/*} </asignatura> }
    </departamento>
} </universidad-1>
```

23.4.3.5. Funciones y tipos

XQuery proporciona una serie de funciones predefinidas, como funciones numéricas y de comparación y manipulación de cadenas. Además, XQuery soporta funciones definidas por el usuario. La siguiente función definida por el usuario devuelve una lista de todas las asignaturas ofertadas por el departamento al que pertenece el profesor especificado:

```
declare function local:dept_asignaturas($iid as xs:string)
        as element(asignatura)* {
  for $i in /universidad/profesor[IID = $iid],
    $c in /universidad/asignatura[nombre_dept = $i/nombre_dept]
  return $c
}
```

El prefijo xs: de espacio de nombres usado en el ejemplo anterior está asociado con el espacio de nombres de XML Schema de forma predefinida en XQuery, mientras que el espacio de nombre local: está asociado de forma predefinida con las funciones locales de XQuery.

Las especificaciones de tipos para los argumentos de la función y los valores devueltos son opcionales y se pueden omitir. XQuery utiliza el sistema de tipo de XML Schema. El tipo element permite elementos con cualquier etiqueta, mientras que element(asignatura) permite elementos con la etiqueta asignatura. Se puede añadir a los tipos * como sufijo para indicar una secuencia de valores de ese tipo; por ejemplo, la definición de la función dept_asignaturas especifica su valor devuelto como una secuencia de elementos asignatura.

La siguiente consulta, que muestra la invocación de la función, escribe las asignaturas del departamento del profesor cuyo nombre es Srinivasan.

```
for $i in /universidad/profesor[nombre = «Srinivasan»],
return local:inst_dept_asignaturas($i/IID)
```

XQuery realiza automáticamente la conversión de tipos siempre que sea necesario. Por ejemplo, si un valor numérico representado por una cadena se compara con un tipo numérico, la conversión de tipo de cadena a número se hace automáticamente. Cuando se pasa un elemento a una función que espera una cadena, la conversión a cadena se hace concatenando todos los valores de texto contenidos (anidados) dentro del elemento. Por tanto, la función contains(a,b), que comprueba si la cadena a contiene a la cadena b, se puede usar con un elemento en su primer argumento, en cuyo caso comprueba si el elemento a contiene la cadena b anidada en cualquier lugar dentro de él. XQuery también proporciona funciones para convertir tipos. Por ejemplo, number(x) convierte una cadena en un número.

23.4.3.6. Otras características

XQuery ofrece una gran variedad de características adicionales, tales como constructores if-then-else, que se pueden usar con cláusulas **return,** y la cuantificación existencial y universal, que se pueden usar en predicados en cláusulas **where.**

Por ejemplo, la cuantificación existencial se puede expresar en la cláusula **where** usando:

```
some $e in ruta satisfies P
```

donde ruta es una expresión de ruta y P es un predicado que puede usar $e. La cuantificación universal se puede expresar usando **every** en lugar de **some.**

Por ejemplo, para encontrar los departamentos en los que todos los profesores tienen un suelo superior a 50.000 €, se puede utilizar la siguiente consulta:

```
for $d in /universidad/departamento
where every $i in /universidad/profesor[nombre_dept =
                $d/nombre_dept]
  satisfies $i/sueldo > 50000
return $d
```

Observe, sin embargo, que si un departamento no tiene ningún profesor, de forma trivial satisface la condición anterior.

La cláusula extra:

```
and fn:exists(/universidad/profesor[nombre_dept =
                $d/nombre_dept])
```

se puede utilizar para asegurar que hay al menos un profesor en el departamento. La función predefinida exists() de la cláusula devuelve cierto si su argumento de entrada no está vacío.

La norma **XQJ** proporciona una API para ejecutar consultas XQuery en un sistema de bases de datos XML y para devolver los resultados XML. Su funcionalidad es similar a la API JDBC.

23.5. La interfaz de programación de aplicaciones de XML

Debido a la gran aceptación de XML como una representación de datos y un formato de intercambio existen una gran cantidad de herramientas de software disponibles para la manipulación de datos XML. Hay dos modelos normalizados para la manipulación de XML por progratma, cada uno disponible para su uso con distintos lenguajes de programación populares. Ambas API se pueden usar para analizar documentos XML y crear su representación en la memoria. Se emplean para aplicaciones que manejan documentos XML individuales. Observe, sin embargo, que no son adecuadas para consultar grandes colecciones de datos XML; los mecanismos de consulta declarativos, tales como XPath y XQuery, son mucho más adecuados para esta tarea.

Una de las API estándar para la manipulación de XML se basa en el *modelo de objetos documento* (Document Object Model, DOM), que trata el contenido XML como un árbol, con cada elemento representado por un nodo, denominado DOMNode. Los programas pueden acceder a partes del documento mediante navegación comenzando con la raíz.

Se encuentran disponibles bibliotecas DOM para la mayor parte de los lenguajes de programación más comunes y están incluso presentes en los exploradores web, en los que se pueden usar para manipular el documento mostrado al usuario. Aquí se explican algunas de las interfaces y métodos de la API DOM de Java, para mostrar cómo puede ser un DOM.

- La API DOM de Java proporciona una interfaz denominada Node e interfaces Element y Attribute, que heredan de la interfaz Node.
- La interfaz Node tiene métodos como getParentNode(), getFirstChild() y getNextSibling() para navegar por el árbol DOM comenzando por el nodo raíz.
- Se puede acceder a los subelementos de un elemento mediante el nombre getElementsByTagName(nombre), que devuelve una lista de todos los elementos hijo con un nombre de etiqueta especificado; se puede acceder a cada miembro de la lista mediante el método item(i), que devuelve el *i*-ésimo elemento de la lista.
- Se puede acceder al valor de un atributo de un elemento por su nombre, usando el método getAttribute(nombre).
- El valor de texto de un elemento se modela como nodo Text, que es un hijo del nodo elemento; un nodo elemento sin subelementos solamente tiene un nodo hijo. El método getData() del nodo Text devuelve el contenido de texto.

DOM también proporciona una serie de funciones para actualizar el documento mediante la adición y el borrado de elementos hijos y atributos, el establecimiento de valores de nodos, etc.

Se necesitan muchos más detalles para escribir un programa DOM real; consulte las notas bibliográficas para obtener más información.

DOM se puede utilizar para acceder a los datos XML almacenados en las bases de datos y se puede construir una base de datos XML con DOM como interfaz principal para acceder y modificar los datos. Sin embargo, la interfaz DOM no soporta ninguna forma de consulta declarativa.

La segunda interfaz de programación empleada más habitualmente, *API simple para XML* (Simple API for XML, SAX), es un modelo de *eventos* diseñado para proporcionar una interfaz común entre analizadores y aplicaciones. Esta API está construida bajo la noción de *manejadores de eventos*, que consisten en funciones especificadas por el usuario asociadas con eventos de análisis. Los eventos de análisis se corresponden con el reconocimiento de partes de un documento; por ejemplo, cuando se encuentra la etiqueta de inicio de un elemento se genera un evento y, cuando se encuentra su etiqueta de finalización, se genera otro. Los fragmentos de un documento siempre se encuentran en orden desde el inicio al final.

El desarrollador de aplicaciones SAX crea funciones manejadoras para cada evento y las registra. Cuando el analizador SAX lee un documento y se produce un evento se llama a la función manejadora con parámetros que describen el evento (como la etiqueta del elemento o el contenido de texto). Las funciones realizan entonces su función. Por ejemplo, para construir un árbol que represente los datos XML, las funciones controladoras para un evento de inicio de un atributo o elemento establecerían el nuevo elemento como el nodo en el que se deben añadir otros nodos hijos. El elemento y el evento correspondientes establecerían el padre del nodo como el nodo actual en el que se pueden añadir más nodos hijos.

SAX requiere generalmente más esfuerzo de programación que DOM, pero ayuda a evitar la sobrecarga de la creación de un árbol DOM en situaciones en las que la aplicación necesita su propia representación de datos. Si se usase DOM en tales aplicaciones, habría sobrecargas innecesarias de espacio y tiempo para la construcción del árbol DOM.

23.6. Almacenamiento de datos XML

Muchas aplicaciones requieren almacenar datos XML. Una forma de almacenar datos XML es como documentos en un sistema de archivos, y otra consiste en construir una base de datos de propósito especial para almacenar datos XML. Otro enfoque se basa en convertir los datos XML a una representación relacional y almacenarla en la base de datos relacional. En esta sección se muestran brevemente varias alternativas para almacenar datos XML.

23.6.1. Almacenamiento de datos no relacionales

Existen varias alternativas para almacenar datos XML en sistemas de almacenamiento de datos no relacionales:

- **Almacenamiento en archivos planos.** Puesto que XML es principalmente un formato de archivo, un mecanismo de almacenamiento natural es simplemente un archivo plano. Este enfoque tiene muchos de los inconvenientes mostrados en el Capítulo 1 sobre el uso de sistemas de archivos como base para las aplicaciones de bases de datos. En particular hay una carencia de aislamiento de datos, comprobaciones de integridad, atomicidad, acceso concurrente y seguridad. Sin embargo, la amplia disponibilidad de herramientas XML que funcionan sobre archivos de datos hace relativamente sencillo el acceso y la consulta de datos XML almacenados en archivos. Por ello, este formato de almacenamiento puede ser suficiente para algunas aplicaciones.
- **Creación de bases de datos XML.** Las bases de datos XML son bases de datos que usan XML como modelo de datos básico. Las primeras bases de datos XML implementaban el modelo de objetos de documento sobre bases de datos orientadas a objetos basadas en C++. Esto permite reutilizar gran parte de la infraestructura de bases de datos orientada a objetos a la vez que se utiliza una interfaz XML estándar. La adición de un XQuery o de otros lenguajes de consultas XML permite consultas declarativas. Otras implementaciones han construido toda la infraestructura de almacenamiento y de consulta XML sobre un gestor de almacenamiento que proporciona soporte transaccional.

Aunque se han creado varios sistemas de bases de datos diseñados específicamente para almacenar datos XML, el desarrollo de un sistema gestor de bases de datos desde cero es una tarea muy compleja. Se debería dar soporte no solo al almacenamiento y consulta de datos XML, sino también a otras características como transacciones, seguridad, soporte de acceso a datos desde clientes

y capacidades de administración. Parece entonces razonable usar un sistema de bases de datos existente para proporcionar estas características e implementar el almacenamiento y la consulta XML, sobre la abstracción relacional, o como una capa paralela a ella. Estas alternativas se estudian en la Sección 23.6.2.

23.6.2. Bases de datos relacionales

Puesto que las bases de datos relacionales se usan ampliamente en aplicaciones existentes, es una gran ventaja almacenar datos XML en bases de datos relacionales de forma que se pueda acceder a los datos desde aplicaciones existentes.

La conversión de datos XML a una forma relacional es normalmente muy sencilla si los datos se han generado en un principio desde un esquema relacional y XML se usó simplemente como un formato de intercambio de datos. Sin embargo, hay muchas aplicaciones en las que los datos XML no se han generado desde un esquema relacional y la traducción de estos a una forma relacional de almacenamiento puede no ser tan sencilla. En particular, los elementos anidados y los elementos que se repiten (correspondientes a atributos con valores de conjunto) complican el almacenamiento de los datos XML en un formato relacional. Se encuentran disponibles diversas alternativas que se describen a continuación.

23.6.2.1. Almacenamiento como cadenas

Los documentos pequeños de XML se pueden almacenar como valores cadenas **(clob)** en tuplas de una base de datos relacional. Los documentos XML grandes cuyo elemento de nivel superior tenga muchos hijos se pueden tratar almacenando cada elemento hijo como una cadena en una tupla distinta de la base de datos. Por ejemplo, los datos XML de la Figura 23.1 se podrían almacenar como un conjunto de tuplas en una relación *elementos*(*datos*), en la que el atributo *datos* de cada tupla almacena un elemento XML (departamento, asignatura, profesor o enseña) en forma de cadena.

Aunque esta representación es fácil de usar, el sistema de la base de datos no conoce el esquema de los elementos almacenados. Por tanto, no es posible consultar los datos directamente. En realidad no es siquiera posible implementar selecciones sencillas, tales como buscar todos los elementos departamento o encontrar el elemento departamento cuyo nombre de departamento sea «Informática», sin explorar todas las tuplas de la relación y examinar los contenidos de la cadena.

Una solución parcial a este problema es almacenar distintos tipos de elementos en relaciones diferentes y también almacenar los valores de algunos elementos críticos como atributos de la relación que permita la indexación. Así, en nuestro ejemplo, las relaciones serían *elementos_departamento*, *elementos_asignatura*, *elementos_profesor* y *elementos_enseña*, cada una con un atributo *datos*. Cada relación puede tener atributos extra para almacenar los valores de algunos subelementos, tales como *nombre_departamento* o *asignatura_id* o *nombre*. Por ello, con esta representación se puede responder eficientemente a una consulta que requiera los elementos departamento con un nombre dado. Este enfoque depende del tipo de información sobre los datos XML, tales como la DTD de los datos.

Algunos sistemas de bases de datos, tales como Oracle, soportan **índices de función** que pueden ayudar a evitar la duplicación de atributos entre la cadena XML y los atributos de la relación. A diferencia de los índices normales, que se construyen sobre los valores de atributos, los índices de función se pueden construir sobre el resultado de aplicar funciones definidas por el usuario a las tuplas. Por ejemplo, se puede construir un índice de función sobre una función definida por el usuario que devuelve el valor del subelemento nombre_dept de la cadena XML en una tupla. El índice se puede entonces usar de la misma forma que un índice sobre un atributo para *nombre_dept*.

Estos enfoques tienen el inconveniente de que una gran parte de la información XML se almacena en cadenas. Es posible almacenar toda la información en relaciones en alguna de las formas que se examinan a continuación.

23.6.2.2. Representación con árboles

Cualquier dato XML se puede modelar como árbol y almacenar mediante el uso de una relación:

nodos(*id*, *id_padre*, *tipo*, *etiqueta*, *vtalor*)

A cada elemento y atributo de los datos XML se le proporciona un identificador único. Una tupla insertada en la relación *nodos* para cada elemento y atributo con su identificador (*id*), el identificador de su nodo padre (*id_padre*), el tipo del nodo (atributo o elemento), el nombre del elemento o atributo (*etiqueta*) y el valor textual del elemento o atributo (*valor*).

Si se debe conservar la información de orden de los elementos y atributos, se puede agregar un atributo adicional, *posición*, a la relación *nodos* para indicar su posición relativa entre los hijos. Como ejercicio intente representar los datos XML de la Figura 23.1 utilizando esta técnica.

Esta representación tiene la ventaja de que toda la información XML se puede representar directamente de forma relacional y se pueden trasladar muchas consultas XML a consultas relacionales y ejecutarlas dentro del sistema de la base de datos. Sin embargo, tiene el inconveniente de que cada elemento se divide en muchas piezas y se necesita un gran número de combinaciones para volver a ensamblar los subelementos en un solo elemento.

23.6.2.3. Representación con relaciones

Según este enfoque, los elementos XML cuyo esquema es conocido se representan mediante relaciones y atributos. Los elementos cuyo esquema es desconocido se almacenan como cadenas o como una representación en árbol.

Se crea una relación para cada tipo de elemento (incluyendo subelementos) cuyo esquema sea conocido y cuyo tipo sea complejo (es decir, que contenga atributos o subelementos). Los atributos de la relación se definen de la siguiente forma:

- Todos los atributos de estos elementos se almacenan como atributos de tipo cadena de la relación.
- Si un subelemento del elemento es de tipo simple (es decir, no contiene atributos o subelementos), se añade un atributo a la relación para representarlo. El tipo predeterminado del atributo es cadena, pero si el subelemento tuviese el tipo XML Schema, se podría usar el tipo SQL correspondiente.

  Por ejemplo, cuando se aplica al elemento *departamento* en el esquema (DTD o XML Schema) de los datos de la Figura 23.1, los subelementos *nombre_dept*, *edificio* y *presupuesto* del elemento *departamento* se convierten en atributos de la relación *departamento*. Aplicando este procedimiento al resto de elementos, se vuelve al esquema relacional original que se ha utilizado en capítulos anteriores.
- En otro caso, se crea una relación correspondiente al subelemento (usando recursivamente las mismas reglas en sus subelementos). Además:
  - Se añade un atributo identificador a las relaciones que representen al elemento (el atributo identificador se añade solo una vez aunque el elemento tenga varios subelementos).
  - Se añade el atributo *id_padre* a la relación que representa al subelemento, almacenando el identificador de su elemento padre.

– Si es necesario conservar el orden, se añade el atributo *posición* a la relación que representa al subelemento.

Por ejemplo, si se aplica este procedimiento al esquema de los datos de la Figura 23.4, se obtienen las siguientes relaciones:

departamento(*id*, *nombre_dept*, *edificio*, *presupuesto*)
asignatura(*id_padre*, *asignatura_id*, *nombre_dept*, *nombre_asig*, *créditos*)

Este enfoque admite diferentes variantes. Por ejemplo, las relaciones correspondientes a subelementos que pueden aparecer solo una vez se pueden «aplanar» en la relación padre trasladándole todos los atributos. En las notas bibliográficas puede encontrar referencias a enfoques diferentes para representar datos XML como relaciones.

23.6.2.4. Publicación y fragmentación de datos XML

Cuando se usa XML para intercambiar datos entre aplicaciones de negocio, los datos muy frecuentemente se originan en bases de datos relacionales. Los datos en las bases de datos relacionales deben ser *publicados*, es decir, convertidos a formato XML para su exportación a otras aplicaciones. Los datos de entrada deben ser *fragmentados*; es decir, convertidos de XML a un formato normalizado de relación, y almacenados en una base de datos relacional. Aunque el código de la aplicación pueda ejecutar las operaciones de publicación y fragmentación, las operaciones son tan comunes que la conversión se debería realizar de forma automática, sin escribir ningún código en la aplicación, siempre que sea posible. Por ello, los fabricantes de bases de datos están trabajando para dar *capacidades XML* a sus productos de bases de datos.

Una base de datos con capacidades XML dispone de mecanismos de publicación de datos relacionales como XML. Esta correspondencia para la publicación de los datos puede ser sencilla o compleja. Una correspondencia sencilla podría asignar un elemento XML a cada fila de una tabla y hacer de cada columna un subelemento del elemento XML. El esquema XML de la Figura 23.1 se puede crear de una representación relacional de información de una universidad usando dicha correspondencia. Esta correspondencia es sencilla de generar automáticamente. Esta vista XML de los datos relacionales se puede tratar como un documento XML *virtual*, y las consultas XML se pueden ejecutar en el documento XML *virtual*.

Una correspondencia más complicada permitiría que se crearan estructuras anidadas. Las extensiones de SQL con consultas anidadas en la cláusula **select** se han desarrollado para permitir una creación fácil de salida de XML anidado. Estas extensiones se describen en la Sección 23.6.3.

También se han definido correspondencias para fragmentar datos XML en una representación relacional. Para los datos XML creados de una representación relacional, la correspondencia requerida para fragmentar los datos es sencilla e inversa a la correspondencia usada para publicar los datos. Para el caso general, se puede generar la correspondencia como se describe en la Sección 23.6.2.3.

23.6.2.5. Almacenamiento nativo en bases de datos relacionales

Algunas bases de datos permiten el **almacenamiento nativo** de XML. Estos sistemas almacenan los datos XML como cadenas o en representaciones binarias más eficientes, sin convertir los datos a la forma relacional. Se introduce el nuevo tipo de datos **xml** para representar datos XML, aunque los tipos CLOB y BLOB pueden proporcionar el mecanismo subyacente de almacenamiento. Se utilizan los lenguajes de consultas XML, como XPath y XQuery, para las consultas de datos XML.

Se puede utilizar una relación con un atributo de tipo **xml** para almacenar una colección de documentos XML; cada documento se almacena como un valor de tipo **xml** en una tupla diferente. Se crean índices ad hoc para indexar los datos XML.

Varios sistemas de bases de datos proporcionan soporte nativo para datos XML. Ofrecen un tipo de datos **xml** y permiten que las consultas XQuery se incorporen dentro de las consultas SQL. Cada consulta XQuery se puede ejecutar en un solo documento XML e incorporar dentro de una consulta SQL para permitir que se ejecute en cada colección de documentos, con cada documento almacenado en una tupla diferente. Por ejemplo, consulte la Sección 30.11 para obtener más detalles sobre el soporte nativo de XML en SQL Server 2005 de Microsoft.

23.6.3. SQL/XML

Aunque XML se utiliza extensamente para el intercambio de datos, los datos estructurados siguen guardándose en bases de datos relacionales. Existe la necesidad habitual de convertir los datos relacionales en representación XML. La norma SQL/XML se ha desarrollado para cubrir esa necesidad; define una extensión normalizada de SQL que permite la creación de respuesta XML anidada. La norma consta de varias partes, incluyendo una manera estándar de identificar los tipos SQL con los tipos de XML Schema, y una manera estándar de identificar esquemas relacionales con esquemas XML, así como extensiones del lenguaje de consultas SQL.

Por ejemplo, la representación SQL/XML de la relación *departamento* tendría un esquema XML con *departamento* como elemento más externo, con cada tupla asociada a un elemento *fila* de XML, y cada atributo de la relación asociada a un elemento de XML del mismo nombre (con algunos convenios para resolver incompatibilidades con los caracteres especiales de los nombres). También se puede asociar un esquema SQL completo, con varias relaciones, a XML de manera parecida. La Figura 23.13 muestra la representación SQL/XML de (parte de) los datos de la *universidad* de la Figura 23.1 que contiene las relaciones *departamento* y *asignatura*.

SQL/XML añade varios operadores y operaciones de agregación a SQL para permitir la construcción de la salida XML directamente a partir de SQL extendido. La función **xmlelement** se puede utilizar para crear elementos de XML, mientras que se puede usar **xmlattributes** para crear atributos, como se muestra en la siguiente consulta:

select xmlelement (**name** «asignatura»,
  xmlattributes (*asignatura_id* **as** *asignatura_id*,
    nombre_dept **as** *nombre_dept*),
  xmlelement (**name** «nombre_asig», *nombre_asig*),
  xmlelement (**name** «créditos», *créditos*))
from *asignatura*

Esta consulta crea un elemento XML para cada asignatura, con el asignatura_id y nombre departamento como atributos, y el nombre de la asignatura y los créditos como subelementos. El resultado sería como los elementos asignatura de la Figura 23.10, pero sin el atributo profesor. El operador **xmlattributes** crea el nombre de atributo XML usando el nombre del atributo SQL, que se puede cambiar usando la cláusula **as,** según se ha visto.

El operador **xmlforest** simplifica la construcción de las estructuras XML. Su sintaxis y comportamiento son similares a los de **xmlattributes,** excepto que crea un bosque (colección) de subelementos, en vez de una lista de atributos. Toma varios argumentos, creando un elemento para cada argumento con el nombre SQL del atributo usado como nombre del elemento XML. El operador del **xmlconcat** se puede utilizar para concatenar los elementos creados por subexpresiones en un bosque.

Cuando el valor SQL que se utiliza para construir un atributo es nulo, se omite. Los valores nulos se omiten cuando se construye el cuerpo de un elemento.

SQL/XML también proporciona la nueva función de agregación **xmlagg,** que crea un bosque (colección) de elementos XML a partir de la colección de los valores a los que se aplica. La siguiente consulta crea un elemento para cada departamento con una asignatura, conteniendo como subelementos todas las asignaturas de dicho departamento. Como la consulta tiene una cláusula **group by** *nombre_dept*, la función de agregación se aplica a todas las asignaturas de todos los departamentos, lo que crea una secuencia de elementos *asignatura_id*.

select xmlelement (**name** «departamento»,
  nombre_dept,
  xmlagg (**xmlforest**(*asignatura_id*)
    order by *asignatura_id*))
from *asignatura*
group by *nombre_dept*

SQL/XML permite que la secuencia creada por **xmlagg** esté ordenada, como se muestra en la consulta anterior. Consulte las notas bibliográficas para más información sobre SQL/XML.

23.7. Aplicaciones XML

A continuación se describen varias aplicaciones de XML para el almacenamiento y comunicación (intercambio) de datos para el acceso a servicios web (recursos de información).

```
<universidad>
  <departamento>
    <fila>
      <nombre_dept> Informática </nombre_dept>
      <edificio> Taylor </edificio>
      <presupuesto> 100000 </presupuesto>
    </fila>
    <fila>
      <nombre_dept> Biología </nombre_dept>
      <edificio> Watson </edificio>
      <presupuesto> 90000 </presupuesto>
    </fila>
  </departamento>
  <asignatura>
    <fila>
      <asignatura_id> CS-101 </asignatura_id>
      <nombre_asig> Intro. a la Informática </nombre_asig>
      <nombre_dept> Informática </nombre_dept>
      <créditos> 4 </créditos>
    </fila>
    <fila>
      <asignatura_id> BIO-301 </asignatura_id>
      <nombre_asig> Genética </nombre_asig>
      <nombre_dept> Biología </nombre_dept>
      <créditos> 4 </créditos>
    </fila>
  <asignatura>
</universidad>
```

Figura 23.13. Representación SQL/XML de (parte de) la información de una universidad.

23.7.1. Almacenamiento de datos con estructura compleja

Muchas aplicaciones necesitan almacenar datos estructurados, pero no se modelan fácilmente como relaciones. Considere, por ejemplo, las preferencias de usuario que debe almacenar una aplicación como un navegador. Normalmente existe un gran número de campos, como la página de inicio y los ajustes de seguridad, de idioma y de configuración de pantalla, que se deben guardar. Algunos de los campos son multivalorados, por ejemplo, una lista de sitios de confianza o, quizás, listas ordenadas, como una lista de marcadores. Las aplicaciones han usado tradicionalmente algún tipo de representación textual para almacenar estos datos. Hoy, un gran número de estas aplicaciones prefieren almacenar esta información de configuración en formato XML. Las representaciones textuales ad hoc utilizadas anteriormente requieren el esfuerzo de diseñar y crear los programas de análisis para leer el archivo y convertir los datos en una forma que el programa pueda utilizar. La representación XML evita estos dos pasos.

Las representaciones basadas en XML se han propuesto incluso como normas para almacenar los documentos, los datos de hojas de cálculo y otros datos que son parte de paquetes de aplicaciones de oficina. El *Formato abierto de documentos* (*ODF–Open Document Format*), que soporta el paquete de software Open Office así como otros paquetes ofimáticos, y el formato *Office Open XML* (*OOXML*), que soporta el paquete Microsoft Office, son normas de representación de documentos basadas en XML. Son los dos formatos más utilizados para la representación de documentos editables.

XML también se usa para representar datos de estructura compleja que se deben intercambiar entre diversas partes de una aplicación. Por ejemplo, un sistema de bases de datos puede representar planes de ejecución de consultas (una expresión del álgebra relacional con información adicional sobre la forma en que se deben ejecutar las operaciones) usando XML. Esto permite que una parte del sistema genere el plan de ejecución de la consulta y otra parte lo muestre sin usar una estructura de datos compartida. Por ejemplo, los datos se pueden generar en un sistema servidor y enviarse a un sistema cliente en el que se muestren.

23.7.2. Formatos normalizados para el intercambio de datos

Se han desarrollado normas basadas en XML para la representación de datos de una gran variedad de aplicaciones especializadas que van desde aplicaciones de negocios, tales como banca y transportes, a aplicaciones científicas, tales como química y biología molecular. Veamos algunos ejemplos:

- La industria química necesita información sobre los compuestos químicos, tales como su estructura molecular, y una serie de propiedades importantes, como los puntos de fusión y ebullición, los valores caloríficos y la solubilidad en distintos disolventes. *ChemML* es una norma para representar dicha información.
- En el transporte, los transportistas de mercancías y los agentes e inspectores de aduana necesitan que los registros de los envíos contengan información detallada sobre los bienes que están siendo transportados, por quién y desde dónde se han enviado, a quién y a dónde se envían, el valor monetario de los bienes, etc.
- Un mercado en red en el que los negocios pueden vender y comprar bienes (el llamado mercado B2B, business-to-business, entre negocios) requiere información tal como los catálogos de los productos, incluyendo descripciones detalladas de estos e información de precios, inventarios, cuotas para una venta propuesta y pedidos de compras. Por ejemplo, las normas *RosettaNet* para aplicaciones de negocios electrónicos definen los esquemas XML y la semántica para la representación de datos, así como normas para el intercambio de mensajes.

El uso de esquemas relacionales normalizados para modelar requisitos de datos tan complejos generaría un gran número de relaciones que no se corresponden directamente con los objetos que se modelan. Las relaciones frecuentemente tienen un gran número de atributos; la representación explícita de nombres de atributos y elementos con sus valores en XML evita confusiones entre los atributos. Las representaciones de elementos anidados ayudan a reducir el número de relaciones que se deben representar, así como el número de combinaciones requeridas para obtener la información, con el posible coste de redundancia. Por ejemplo, en nuestro ejemplo de la universidad el listado de departamentos con elementos asignatura anidados en elementos departamento, como en la Figura 23.4, da lugar a un formato más adecuado para algunas aplicaciones —en particular para su legibilidad— que la representación normalizada de la Figura 23.1.

23.7.3. Servicios Web

Las aplicaciones a menudo requieren datos externos a la organización o de otro departamento de la misma empresa que usa una base de datos diferente. En muchas de estas situaciones la empresa externa o el departamento no están dispuestos a permitir el acceso directo a su base de datos usando SQL, sino solo a proporcionar información limitada a través de interfaces predefinidas.

Cuando son las personas quienes deben usar directamente la información, las empresas proporcionan formularios web en los que los usuarios pueden introducir valores y conseguir la información deseada en HTML. Sin embargo, hay muchas aplicaciones en las que son los programas los que acceden a esta información, en lugar de las personas. Proporcionar los resultados de una consulta en XML es un requisito claro. Además, tiene sentido especificar los valores de la entrada a la consulta también en formato XML.

En efecto, el proveedor de información define los procedimientos cuya entrada y salida están en formato XML. El protocolo HTTP se utiliza para comunicar la información de entrada y salida, puesto que es muy utilizado y puede pasar a través de los cortafuegos que las instituciones emplean para mantener aislado el tráfico no deseado de Internet.

El **protocolo simple de acceso a objetos** (SOAP: *simple object access protocol*) define una norma para invocar procedimientos usando XML para representar la entrada y salida de los procedimientos. SOAP define un esquema XML estándar para representar el nombre del procedimiento y de los indicadores de estado del resultado, como fallo y error. Los parámetros y resultados de los procedimientos son datos XML dependientes de las aplicaciones incorporados en las cabeceras XML de SOAP.

Normalmente se usa HTTP como protocolo de transporte para SOAP, pero también se puede emplear un protocolo basado en mensajes (como correo electrónico sobre el protocolo SMTP). Actualmente la norma SOAP se utiliza mucho. Por ejemplo, Amazon y Google proporcionan procedimientos basados en SOAP para realizar búsquedas y otras actividades. Otras aplicaciones que proporcionen servicios de alto nivel a los usuarios pueden invocar a estos procedimientos. La norma SOAP es independiente del lenguaje de programación subyacente, y es posible que un sitio que funciona con un lenguaje, como C#, invoque un servicio que funcione en otro lenguaje, como Java.

Un sitio que proporcione tal colección de procedimientos SOAP se denomina **servicio web.** Se han definido varias normas para dar soporte a los servicios web. El **lenguaje de descripción de servicios web** (WSDL: *web services description language*) es un lenguaje usado para describir las capacidades de los servicios web. WSDL proporciona las capacidades que la definición de la interfaz (o las definiciones de las funciones) proporciona en un lenguaje de programación tradicional, especificando las funciones disponibles y sus tipos de entrada y de salida. Además, WSDL permite la especificación de la URL y del número de puerto de red que se utilizarán para invocar el servicio web. Hay también una norma denominada **descripción, descubrimiento e integración universales** (UDDI: *universal description, discovery and integration*), que define la forma en que se puede crear un directorio de los servicios web disponibles y cómo un programa puede buscar en el directorio para encontrar un servicio web que satisfaga sus necesidades.

El siguiente ejemplo muestra el valor de los servicios web. Una compañía de líneas aéreas puede definir un servicio web que proporcione el conjunto de procedimientos que un sitio web de una agencia de viajes puede invocar; puede incluir procedimientos para encontrar horarios de vuelo y la información de tasas, así como para hacer reservas. El sitio web de la agencia puede interactuar con varios servicios web proporcionados por diversas líneas aéreas, hoteles y otras compañías, ofrecer la información del viaje a los clientes y hacer reservas. Al dar soporte a los servicios web, las empresas permiten que se construya un servicio útil que integre servicios individuales. Los usuarios pueden interactuar con un solo sitio web para hacer sus reservas sin tener que entrar en contacto con varios sitios web diferentes.

Para invocar un servicio web, los clientes deben preparar un mensaje XML apropiado bajo SOAP y enviarlo al servicio; cuando consigue el resultado en XML, el cliente debe extraer entonces la información del resultado XML. Existen algunas API estándar en lenguajes como Java y C# para crear y extraer la información de los mensajes SOAP.

Consulte las notas bibliográficas para más información sobre los servicios web.

23.7.4. Mediación de datos

La compra comparada es un ejemplo de aplicación de mediación, en la que los datos sobre elementos, inventario, precio y costes de envío se extraen de una serie de sitios web que ofrecen un elemento concreto en venta. La información agregada resultante es significativamente más valiosa que la información individual ofrecida por un único sitio.

Un gestor financiero personal es una aplicación similar en el contexto de la banca. Considere un consumidor con una gran cantidad de cuentas a gestionar, tales como cuentas bancarias, cuentas de ahorro y cuentas de jubilación. Suponga que estas cuentas pueden estar en distintas instituciones. Es un reto importante proporcionar una gestión centralizada de todas las cuentas de un cliente. La mediación basada en XML soluciona el problema extrayendo una representación XML de la información de la cuenta desde los sitios web respectivos de las instituciones financieras en las que está cada cuenta. Esta información se puede extraer fácilmente si la institución la exporta a un formato XML estándar, por ejemplo, como servicio web. Para aquellas que no lo hacen se usa un software *envolvente* (*wrapper*) para generar datos XML a partir de las páginas web HTML devueltas por el sitio web. Las aplicaciones envolventes necesitan un mantenimiento constante puesto que dependen de los detalles de formato de las páginas web, que cambian constantemente. No obstante, el valor proporcionado por la mediación frecuentemente justifica el esfuerzo requerido para desarrollar y mantener las aplicaciones envolventes.

Una vez que se dispone de las herramientas básicas para extraer la información de cada fuente, se usa una aplicación *mediadora* para combinar la información extraída bajo un único esquema. Esto puede requerir más transformación de los datos XML de cada sitio, puesto que los distintos sitios pueden estructurar la misma in-

formación de una forma diferente. Pueden usar nombres diferentes para la misma información (por ejemplo, num_cuenta e id_cuenta), o pueden incluso usar el mismo nombre para información distinta. El mediador debe decidir sobre un único esquema que representa toda la información requerida, y debe proporcionar código para transformar los datos entre diferentes representaciones. Estos temas se tratan con más detalle en la Sección 19.8 en el contexto de las bases de datos distribuidas. Los lenguajes de consultas XML, tales como XSLT y XQuery, desempeñan un papel importante en la tarea de transformación entre distintas representaciones XML.

23.8. Resumen

- Al igual que el lenguaje de marcas de hipertexto HTML (HyperText Markup Language), en que se basa la web, el lenguaje de marcas extensible, XML (Extensible Markup Language), es descendiente del lenguaje estándar generalizado de marcas (SGML, Standard Generalized Markup Language). XML estaba pensado inicialmente para proporcionar marcas funcionales para documentos web, pero se ha convertido de facto en el formato de datos estándar para el intercambio de datos entre aplicaciones.
- Los documentos XML contienen elementos con etiquetas de inicio y finalización correspondientes que indican el comienzo y finalización de un elemento. Los elementos pueden tener subelementos anidados a ellos, hasta cualquier nivel de anidamiento. Los elementos pueden también tener atributos. En el contexto de la representación de datos la elección entre representar información como atributos o como subelementos suele ser arbitraria.
- Los elementos pueden tener un atributo del tipo ID que almacene un identificador único para el elemento. Los elementos también pueden almacenar referencias a otros elementos utilizando atributos del tipo IDREF. Los atributos del tipo IDREF pueden almacenar una lista de referencias.
- Los documentos pueden, opcionalmente, tener su esquema especificado mediante una definición de tipos de documentos (DTD, Document Type Declaration). La DTD de un documento especifica los elementos que pueden aparecer, cómo se pueden anidar y los atributos que puede tener cada elemento. Aunque las DTD se usan mucho, tienen graves limitaciones. Por ejemplo, no proporcionan un sistema de tipos.
- XML Schema es en la actualidad el mecanismo estándar para especificar el esquema de los documentos XML. Proporciona un gran conjunto de tipos básicos, así como constructores para crear tipos complejos y especificar restricciones de integridad, incluidas las restricciones de clave y de claves externas (*keyref*).
- Los datos XML se pueden representar como estructuras en árbol, con nodos correspondientes a los elementos y atributos. El anidamiento de elementos se refleja mediante la estructura padre- hijo de la representación en árbol.
- Se pueden usar expresiones de ruta para recorrer la estructura de árbol XML y así localizar los datos requeridos. XPath es un lenguaje normalizado para las expresiones de rutas y permite especificar los elementos necesarios mediante una ruta parecida a un sistema de archivos y, además, la selección y otras características. XPath también forma parte de otros lenguajes de consultas XML.
- El lenguaje XQuery es el estándar para la consulta de datos XML. Tiene una estructura similar a la de SQL, con cláusulas **for, let, where, order by** y **return.** Sin embargo, dispone de muchas extensiones para tratar con la naturaleza en árbol de XML y para permitir la transformación de documentos XML en otros documentos con una estructura significativamente diferente. Las expresiones de ruta XPath forman parte de XQuery. XQuery soporta consultas anidadas y funciones definidas por el usuario.
- Las API DOM y SAX se usan mucho para el acceso mediante programación a datos XML. Estas API están disponibles para distintos lenguajes de programación.
- Los datos XML se pueden almacenar de varias formas distintas: en sistemas de archivos, o en bases de datos XML, que emplean XML como representación interna.
- XML se puede almacenar como cadenas en una base de datos relacional. Alternativamente, las relaciones pueden representar datos XML como árboles. Otra alternativa es que los datos XML se pueden hacer corresponder con relaciones, de la misma forma que se hacen corresponder los esquemas E-R con esquemas relacionales. El almacenamiento nativo de XML en las bases de datos relacionales se facilita añadiendo el tipo de datos **xml** a SQL.
- XML se utiliza en gran variedad de aplicaciones, como el almacenamiento de datos complejos, el intercambio de datos entre empresas de forma estándar, la mediación de datos y los servicios web. Los servicios web proporcionan una interfaz de llamada a procedimientos remotos, con XML como mecanismo para codificar los parámetros y los resultados.

Términos de repaso

- Lenguaje de marcas extensible (*extensible markup language*: XML).
- Lenguaje de marcas de hipertexto (*hyper-text markup language*: HTML).
- Lenguaje estándar generalizado de marcas (*standard generalized markup language*: SGML).
- Lenguaje de marcas.
- Etiquetas.
- Autodocumentado.
- Elemento.
- Elemento raíz.
- Elementos anidados.
- Atributos.
- Espacio de nombres.
- Espacio de nombres predeterminado.
- Definición de esquema.
- Definición de tipos de documentos (*document type definition*: DTD).
  - ID.
  - IDREF e IDREFS.
- XML Schema.
  - Tipos simples y complejos.
  - Tipo de secuencia.
  - Clave y keyref.
  - Restricciones de aparición.
- Modelo de árbol de datos XML.
- Nodos.

- Consulta y transformación.
- Expresiones de ruta.
- XPath.
- XQuery.
  - Expresiones FLWOR.
    - **for.**
    - **let.**
    - **where.**
    - **order by.**
    - **return.**
  - Reuniones.
  - Expresión FLWOR anidada.
  - Ordenación.
- API XML.
- Modelo de objetos de documento (*document object model*: DOM).
- API simple para XML (*simple API for XML*: SAX).
- Almacenamiento de datos XML.
  - En almacenamientos de datos no relacionales.
  - En bases de datos relacionales.
    - Almacenamiento como cadena.
    - Representación en árbol.
    - Representación con relaciones.
    - Publicación y fragmentación.
    - Base de datos con capacidades XML.
    - Almacenamiento nativo.
    - SQL/XML.
- Aplicaciones XML.
  - Almacenamiento de datos complejos.
  - Intercambio de datos.
  - Mediación de datos.
  - SOAP.
  - Servicios web.

Ejercicios prácticos

23.1. Indique una representación alternativa de la información de la universidad que contenga los mismos datos que la Figura 23.1, pero usando atributos en lugar de subelementos. Escriba también la DTD o el XML Schema para esta representación.

23.2. Escriba la DTD o el XML Schema para una representación XML del siguiente esquema relacional anidado:

$$Emp = (nombre, ConjuntoHijos\ \mathbf{setof}(Hijos),\ ConjuntoMaterias\ \mathbf{setof}(Materias))$$
$$Hijos = (nombre, Cumpleaños)$$
$$Cumpleaños = (día, mes, año)$$
$$Materias = (tipo, ConjuntoExámenes\ \mathbf{setof}(Exámenes))$$
$$Exámenes = (año, ciudad)$$

23.3. Escriba una consulta en XPath sobre el esquema del Ejercicio práctico 23.2 para listar todos los tipos de materia de *Emp*.

23.4. Escriba una consulta en XQuery en la representación XML de la Figura 23.10 para encontrar el sueldo total de todos los profesores de cada uno de los departamentos.

23.5. Escriba una consulta en XQuery en la representación XML de la Figura 23.1 para calcular la reunión externa por la izquierda de los elementos departamento con los elementos asignatura. (Sugerencia: se puede usar la cuantificación universal).

23.6. Escriba consultas en XQuery que devuelvan los elementos departamento con los elementos asignatura asociados anidados en los elementos departamento, dada la representación de la información de universidad usando ID e IDREFS de la Figura 23.10.

23.7. Escriba un esquema relacional para representar la información bibliográfica como se especifica en el fragmento DTD de la Figura 23.14. El esquema relacional debe registrar el orden de los elementos autor. Se puede suponer que solo aparecen como elementos de nivel superior en los documentos XML los libros y los artículos.

23.8. Muestre la representación en árbol de los datos XML de la Figura 23.1 y la representación del árbol usando las relaciones *nodos* e *hijo* descritas en la Sección 23.6.2.

23.9. Considere la siguiente DTD recursiva:

```
<!DOCTYPE partes [
<!ELEMENT producto (nombre, infocomponente*)>
<!ELEMENT infocomponente (producto, cantidad)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT cantidad (#PCDATA)>
] >
```

a. Escriba un pequeño ejemplo de datos correspondientes a esta DTD.

b. Muestre cómo hacer corresponder esta DTD con un esquema relacional. Puede suponer que los nombres de producto son únicos, es decir, cada vez que aparezca un producto, la estructura de sus componentes será la misma.

c. Cree un esquema en XML Schema correspondiente a esta DTD.

```
<!DOCTYPE bibliografía [
  <!ELEMENT libro (título, autor+, año, editor, lugar?)>
  <!ELEMENT artículo (título, autor+, revista, año, número,
                      volumen, páginas?)>
  <!ELEMENT autor (apellidos, nombre) >
  <!ELEMENT título (#PCDATA)>
  ...declaraciones PCDATA similares para el año, el editor, el lugar,
     la revista, el año, el número, el volumen, las páginas,
     los apellidos y el nombre
] >
```

Figura 23.14. DTD para datos bibliográficos.

Ejercicios

23.10. Demuestre, proporcionando una DTD, cómo representar la relación *libros*, que no está en primera forma normal, de la Sección 22.2 mediante el uso de XML.

23.11. Escriba las siguientes consultas en XQuery, usando la DTD del Ejercicio práctico 23.2.

a. Encontrar los nombres de todos los empleados que tienen un hijo cuyo cumpleaños cae en marzo.

b. Encontrar aquellos empleados que se examinaron del tipo de materia «mecanografía» en la ciudad de «Dayton».

c. Listar todos los tipos de materias de *Emp*.

23.12. Considere los datos XML de la Figura 23.2. Suponga que se desea encontrar los pedidos con dos o más compras del producto con identificador 123. Considere esta posible solución al problema:

```
for $p in pedido_compra
where $p/elemento/identificador =
          123 and $p/elemento/cantidad >= 2
return $p
```

Explique por qué la consulta puede devolver algunos pedidos con menos de dos compras del producto 123. Escriba una versión correcta de esta consulta.

23.13. Escriba una consulta en XQuery para invertir el anidamiento de los datos del Ejercicio 23.10. Es decir, el nivel más externo del anidamiento de la salida debe tener los elementos correspondientes a los autores, y cada uno de estos elementos debe tener anidados los elementos correspondientes a todos los libros escritos por el autor.

23.14. Escriba la DTD de una representación XML de la información de la Figura 7.29. Cree un tipo de elemento diferente para representar cada relación, pero utilice ID e IDREF para implementar las claves primarias y externas.

23.15. Escriba una representación en XML Schema de la DTD del Ejercicio 23.14.

23.16. Escriba las consultas en XQuery del fragmento DTD de bibliografía de la Figura 23.14 para realizar lo siguiente:

a. Determinar todos los autores que tienen un libro y un artículo en el mismo año.

b. Mostrar los libros y artículos ordenados por año.

c. Mostrar los libros con más de un autor.

d. Encontrar todos los libros que contengan la palabra «bases de datos» en su título y la palabra «Hank» en el nombre o apellidos del autor.

23.17. Escriba la versión relacional del esquema de pedidos en XML de la Figura 23.2, usando el enfoque descrito en la Sección 23.6.2.3. Sugiera cómo eliminar la redundancia en el esquema relacional cuando los identificadores determinan funcionalmente su descripción, y los nombres del comprador y del proveedor determinan funcionalmente las direcciones del comprador y del proveedor, respectivamente.

23.18. Escriba consultas en SQL/XML para convertir los datos de la universidad del esquema relacional que hemos utilizado en capítulos anteriores a los esquemas XML *universidad-1* y *universidad-2*.

23.19. Como en el Ejercicio 23.18, escriba consultas para convertir los datos de la universidad a los esquemas XML *universidad-1* y *universidad-2*, pero esta vez escribiendo consultas XQuery sobre la base de datos SQL/XML predeterminada a la versión XML.

23.20. Una forma de fragmentar un documento XML es emplear XQuery para convertir el esquema a la versión SQL/XML del esquema relacional correspondiente y después usar esta versión en sentido inverso para rellenar la relación.

Como ejemplo, indique una consulta XQuery para convertir datos del esquema XML *universidad-1* al esquema SQL/XML de la Figura 23.13.

23.21. Considere el esquema XML de ejemplo de la Sección 23.3.2 y escriba consultas XQuery para realizar las siguientes tareas:

a. Comprobar si se cumple la restricción de clave mostrada en la Sección 23.3.2.

b. Comprobar si se cumple la restricción keyref mostrada en la Sección 23.3.2.

23.22. Considere el Ejercicio práctico 23.7 y suponga que los autores también pueden aparecer como elementos de nivel superior ¿Qué cambio habría que realizar en el esquema relacional?

Herramientas

Se encuentran disponibles en el dominio público una serie de herramientas para tratar con XML. El sitio web del W3C, www.w3.org, tiene páginas que describen las diferentes normas relacionadas con XML, así como punteros a las herramientas de software como implementaciones de lenguajes. En www.w3.org/XML/Query está disponible una extensa lista de implementaciones de XQuery. Saxon D (saxon.sourceforge.net) y Galax (http://www.galaxquery.org/) son útiles como herramientas de aprendizaje, aunque no se han diseñado para manejar grandes bases de datos. Exist (exist-db.org) es una base de datos XML de código abierto, que dispone de varias características. Varias bases de datos comerciales, incluidas DB2 de IBM, Oracle y SQL Server de Microsoft, soportan el almacenamiento en XML, la publicación empleando varias extensiones de SQL, así como las consultas mediante XPath y XQuery.

Notas bibliográficas

El consorcio W3C (World Wide Web Consortium) actúa como cuerpo normativo para las normas relacionadas con la web, incluyendo XML básico y todos los lenguajes relacionados con XML, tales como XPath, XSLT y XQuery. En www.w3.org se pueden obtener gran número de informes técnicos que definen las normas relacionadas con XML. Este sitio también contiene tutoriales y punteros a software que implementan las distintas normas.

El lenguaje XQuery deriva de un lenguaje de consulta de XML llamado Quilt; el propio Quilt incluía funciones de lenguajes anteriores como XPath, tratadas en la Sección 23.4.2, así como otros dos lenguajes de consulta de XML, XQL y XML-QL. Quilt se describe en Chamberlin et ál [2000]. Deutsch et ál. [1999] describen el lenguaje XML-QL. El W3C lanzó una *recomendación candidata* de una extensión de XQuery a mediados de 2009 que incluía actualizaciones.

En el texto Katz et ál. [2004] se proporciona un tratamiento detallado de XQuery. La especificación de XQuery se puede encontrar en www.w3.org/TR/xquery. También están disponibles en el sitio especificaciones de las extensiones de XQuery, incluyendo la funcionalidad XQuery Update y la extensión XQuery Scripting. La integración de consultas con palabras clave en XML se describe en Florescu et ál. [2000] y Amer-Yahia et ál. [2004].

Funderburk et ál. [2002a], Florescu y Kossmann [1999], Kanne y Moerkotte [2000] y Shanmugasundaram et ál. [1999] describen el almacenamiento de datos XML. Eisenberg y Melton [2004a] proporcionan una descripción de SQL/XML, mientras que Funderburk et ál. [2002b] ofrecen descripciones de SQL/XML y de XQuery. Consulte los Capítulos 28 a 30 para más información sobre el soporte de XML en bases de datos comerciales. Eisenberg y Melton [2004b] proporcionan una descripción del API XQJ para XQuery, aunque la definición de la norma se puede encontrar en línea en http://www.w3.org/TR/xquery.

Indexado de XML, procesamiento y optimización de consultas: la indexación de datos XML y el procesamiento y optimización de consultas de XML ha sido un área de gran interés en los últimos años. Se han publicado un gran número de artículos. Uno de los retos de la indexación es que las consultas pueden especificar una selección sobre una ruta, como $/a/b//c[d=«CSE»]$; el índice debe soportar la recuperación eficiente de los nodos que satisfagan la especificación de ruta y la selección de valores. Entre los trabajos sobre indexación de datos XML están los de Pal et ál. [2004] y Kaushik et ál. [2004]. Si los datos se fragmentan y almacenan en relaciones, la evaluación de una expresión de ruta se corresponde con el cálculo de una reunión. Se han propuesto diversas técnicas para la numeración de nodos en los datos de XML, que se puede utilizar para comprobar eficientemente si un nodo es descendiente de otro; consulte, por ejemplo, O'Neil et ál. [2004]. Entre los trabajos sobre optimización de consultas de XML se encuentran los de McHugh y Widom [1999], Wu et ál. [2003] y Krishnaprasad et ál. [2004].

Parte 8
Temas avanzados

El Capítulo 24 trata varios temas avanzados en el desarrollo de aplicaciones empezando con el ajuste de rendimiento de sistemas de bases de datos para aumentar la velocidad de las aplicaciones. También describe las pruebas que se usan para medir el rendimiento de los sistemas de bases de datos comerciales. También se tratan temas del desarrollo de aplicaciones como las pruebas y la migración de aplicaciones. Concluye con una descripción general del proceso de normalización y las normas de lenguajes de bases de datos existentes.

El Capítulo 25 describe los tipos de datos, tales como los datos temporales, los espaciales y los multimedia, y los aspectos de su almacenamiento en bases de datos. Las aplicaciones como la informática móvil también se describen en este capítulo.

Finalmente, en el Capítulo 26 se describen varias técnicas avanzadas de procesamiento de transacciones, entre ellas los monitores de procesamiento de transacciones, los flujos de trabajo transaccionales y el procesamiento de transacciones en el comercio electrónico. El capítulo trata después de los sistemas de bases de datos en memoria principal y los sistemas transaccionales en tiempo real, para concluir con una discusión sobre las transacciones de larga duración.

Desarrollo avanzado de aplicaciones

24

En el desarrollo de aplicaciones se pueden distinguir varias tareas. En los Capítulos 7-9 se estudió la forma de diseñar y desarrollar una aplicación. Uno de los aspectos del diseño de aplicaciones es el rendimiento que se espera de ellas. De hecho, es común encontrar que al finalizar el desarrollo de una aplicación esta funciona más lentamente de lo que se hubiera deseado, o gestiona menos transacciones por segundo de las que se requieren. Las aplicaciones que necesitan una excesiva cantidad de tiempo para realizar las acciones requeridas pueden generar el descontento de los usuarios, en el mejor de los casos, y ser totalmente inutilizable, en el peor.

Se puede hacer que las aplicaciones se ejecuten significativamente más rápido mediante el ajuste del rendimiento, que consiste en buscar y eliminar los cuellos de botella y añadir el hardware adecuado, como puede ser memoria o discos. Los desarrolladores de aplicaciones pueden ajustar las aplicaciones de diversas formas, así como los administradores del sistema de base de datos pueden acelerar el procesamiento de las aplicaciones.

Las pruebas normalizadas son conjuntos programados de tareas que ayudan a caracterizar el rendimiento de un sistema de bases de datos. Son útiles para obtener una idea general de los requisitos de hardware y software de una aplicación, incluso antes de que se construya.

Hay que probar las aplicaciones a la vez que se desarrollan. Las pruebas requieren que se generen estados de la base de datos y entradas de prueba, así como la verificación de que las salidas que generan coinciden con las esperadas. Posteriormente se tratarán estos temas sobre la prueba de aplicaciones. Los sistemas heredados son aplicaciones que han quedado obsoletas, normalmente basadas en tecnologías anticuadas. Sin embargo, suelen formar parte del núcleo de la organización y ejecutar aplicaciones críticas. Se tratarán los temas sobre cómo interactuar con ellas y los procesos de migración, sustituyéndolas por sistemas más modernos.

Las normas son muy importantes para el desarrollo de las aplicaciones, especialmente en la época de Internet, dado que estas necesitan comunicarse entre sí para llevar a cabo tareas útiles. Se han propuesto varias normas que afectan al desarrollo de las aplicaciones de bases de datos.

24.1. Ajuste del rendimiento

El ajuste del rendimiento de un sistema implica ajustar varios parámetros y opciones de diseño para mejorar su rendimiento en una aplicación concreta. Existen varios aspectos del diseño de los sistemas de bases de datos que afectan al rendimiento de las aplicaciones (aspectos de alto nivel, como el esquema y el diseño de las transacciones; los parámetros de las bases de datos, como los tamaños de la memoria intermedia, y los aspectos del hardware, como el número de discos). Cada uno de estos aspectos puede ajustarse de modo que se mejore el rendimiento.

24.1.1. Mejora de la orientación de conjuntos

Cuando se ejecutan las consultas de SQL desde un programa de aplicación suele ocurrir que la consulta se ejecuta con cierta frecuencia, pero con distintos valores de sus parámetros. Cada llamada tiene una sobrecarga de comunicación con el servidor, además de la sobrecarga de procesamiento en el servidor.

Por ejemplo, suponga un programa que recorre los departamentos invocando una consulta de SQL embebida para calcular el sueldo total de todos los profesores de los departamentos:

select sum(*sueldo*)
from *profesor*
where *nombre_dept*=?

Si la relación *profesor* no tiene un índice agrupado por *nombre_dept*, cada una de estas consultas genera un recorrido en toda la relación. Aunque exista este índice, se requiere una operación de E/S aleatoria para cada valor de *nombre_dept*.

En su lugar, se podría utilizar una única consulta de SQL para encontrar los gastos en sueldos de los departamentos:

select *nombre_dept*, **sum**(*sueldo*)
from *profesor*
group by *nombre_dept*;

Esta consulta se puede evaluar con una única exploración de la relación *profesor*, evitando realizar una E/S aleatoria para cada departamento. Los resultados se pueden recoger en el lado cliente usando una única ronda de comunicación y el programa cliente puede recorrer los resultados para encontrar la agregación de los departamentos. Al combinar múltiples consultas de SQL en una única, como anteriormente, en muchos casos se pueden reducir de forma muy importante los costes de ejecución, si la relación *profesor* es muy grande y tiene un gran número de departamentos.

La API de JDBC también proporciona una función denominada **actualización en lotes** que permite realizar un cierto número de inserciones utilizando una única comunicación con la base de datos. En la Figura 24.1 se muestra el uso de esta función. El código de la figura solo requiere una ronda de comunicación con la base de datos, cuando se ejecuta el método executeBatch(), a diferencia del código similar sin la función de actualización en lotes que se vio en la Figura 5.2. Sin la actualización en lotes se necesitan tantas rondas de comunicación con la base de datos como profesores existan. La función de actualización en lotes también permite que la base de datos procese un lote de inserciones de una sola vez, lo que puede ser mucho más eficiente que una serie de inserciones de registros individuales.

Otra técnica muy utilizada en sistemas cliente-servidor para reducir el coste de la comunicación y compilación de SQL es utilizar procedimientos almacenados, en los que las consultas se almace-

nan en el servidor en forma de procedimientos que pueden estar precompilados. Los clientes invocan estos procedimientos almacenados, en lugar de comunicar una serie de consultas.

Otro aspecto de mejora de la orientación de conjuntos consiste en reescribir las consultas con **subconsultas anidadas.** En el pasado, los optimizadores de muchos sistemas de bases de datos no eran particularmente buenos, por lo que la forma de escribir una consulta tenía una gran influencia sobre cómo se ejecutaba y, por tanto, en el rendimiento. En la actualidad, los optimizadores avanzados pueden transformar incluso consultas pobremente escritas y ejecutarlas de forma eficiente, por lo que la necesidad de ajustar las consultas resulta menos importante de lo que solía ser. Sin embargo, en el caso de consultas complejas que contienen subconsultas anidadas no suelen resultar muy optimizadas por muchos de los optimizadores.

Ya se vieron las técnicas para la descorrelación de subconsultas anidadas en la Sección 13.4.4. Si una subconsulta no está descorrelacionada, se ejecuta repetidamente, con el resultado potencial de generar gran aleatoriedad de E/S. En contraste, la descorrelación hace posible operaciones orientadas a conjuntos más eficientes como las reuniones, que permiten minimizar las E/S aleatorias. La mayoría de los optimizadores de consultas de bases de datos incorporan algunas formas de descorrelación, pero algunos solamente manejan subconsultas anidadas muy simples. Se puede obtener el plan de ejecución seleccionado por el optimizador como se describió en el Capítulo 13. Si el optimizador no ha conseguido descorrelacionar una subconsulta anidada, la consulta se puede descorrelacionar reescribiéndola manualmente.

```
PreparedStatement pStmt = conn.prepareStatement(
                          "insert into profesor values(?,?,?,?)");
pStmt.setString(1, "88877");
pStmt.setString(2, "Perry");
pStmt.setInt(3, "Finanzas");
pStmt.setInt(4, 125000);
pStmt.addBatch( );
pStmt.setString(1, "88878");
pStmt.setString(2, "Thierry");
pStmt.setInt(3, "Física");
pStmt.setInt(4, 100000);
pStmt.addBatch( ); pStmt.executeBatch( );
```

Figura 24.1. Actualización en lotes en JDBC.

24.1.2. Ajuste de carga y actualización masivas

Cuando se carga un gran volumen de datos en una base de datos (llamada operación de **carga masiva**), el rendimiento suele ser muy pobre si las inserciones se realizan con sentencias de inserción separadas. Una razón es la sobrecarga que supone analizar cada consulta SQL; una razón más importante es que al realizar las comprobaciones de restricciones de integridad y las actualizaciones de índices por separado para cada tupla que se inserta genera un gran número de operaciones de E/S aleatorias. Si las inserciones se realizan como un gran lote, las comprobaciones de las restricciones de integridad y la actualización de los índices se pueden realizar de forma orientada a conjuntos, reduciendo de forma importante la sobrecarga y aumentando el rendimiento en un orden de magnitud o superior.

Para disponer de operaciones de carga masiva, la mayoría de los sistemas de bases de datos proporcionan la utilidad de **importación masiva** y una utilidad correspondiente de **exportación masiva.** La utilidad de importación masiva lee los datos de un archivo y realiza las comprobaciones de las restricciones de integridad, así como el mantenimiento de los índices de forma muy eficiente. Los formatos de archivos de entrada y salida que permiten estas utilidades de importación/exportación masiva incluyen los archivos de texto con caracteres separadores, como comas o tabuladores, para separar los valores de los atributos, con cada registro en una línea distinta (a estos formatos de archivo se les denomina formatos de *valores separados por comas* o *valores separados por tabuladores*). También suelen permitir formatos binarios específicos, así como formatos en XML. Los nombres de las utilidades de importación/exportación masiva difieren entre las bases de datos. En PostgreSQL, se llaman pg dump y pg restore (PostgreSQL también proporciona el comando de SQL **copy,** que proporciona una funcionalidad similar). La utilidad de importación/exportación masiva de Oracle se denomina **SQL*Loader,** la de DB2 se llama load y la de SQL Server se llama bcp (SQL Server también proporciona un comando de SQL llamado **bulk insert**).

Ahora consideraremos el caso de ajuste de actualizaciones en lotes. Suponga una relación *fondos_recibidos*(*nombre_dept*, *cantidad*) que guarda los fondos recibidos (por ejemplo, mediante una transferencia de fondos) para cada uno de los departamentos. Suponga que se quiere la suma de los saldos correspondientes a los presupuestos de los departamentos. Para llevar a cabo la sentencia de actualización de SQL hay que realizar un recorrido en la relación *fondos_recibidos* para cada tupla de la relación *departamento*. Se pueden utilizar subconsultas en la cláusula de actualización para esta tarea, de la siguiente forma: se supone, por simplificar, que la relación *fondos_recibidos* contiene como mucho una tupla para cada departamento.

```
update departamento set presupuesto = presupuesto +
        (select cantidad
         from fondos_recibidos
         where fondos_recibidos.nombre_dept =
               departamento.nombre_dept)
  where exists(
         select *
          from fondos_recibido
          where fondos_recibidos.nombre_dept =
                departamento.nombre_dept);
```

Obsrve que la condición de la cláusula **where** de la actualización asegura que solo se actualizan las cuentas con tuplas correspondientes de *fondos_recibidos*, mientras que la subconsulta con la cláusula **set** calcula la cantidad que hay que añadir a cada uno de los departamentos.

Existen muchas aplicaciones que requieren una actualización como la que se mostró anteriormente. Normalmente existe una tabla que llamaremos **tabla maestra,** y las actualizaciones en esta tabla maestra se reciben como un lote. Entonces hay que actualizar dicha tabla maestra de la forma correspondiente. SQL:2003 proporciona una construcción especial, llamada **merge,** que simplifica la tarea de realizar dicha mezcla de información. Por ejemplo, la actualización anterior se puede expresar con **merge** de la siguiente forma:

```
merge into departamento as A
using (select *
       from fondos_recibidos) as F
  on (A.nombre_dept = F.nombre_dept)
when matched then
  update set presupuesto = presupuesto + F.cantidad;
```

Cuando un registro de la subconsulta de la cláusula **using** coincide con un registro de la relación *departamento*, se ejecuta la cláusula **when matched,** que puede ejecutar una actualización en la relación; en este caso, el registro coincidente de la relación *departamento* se actualiza como se indica.

La sentencia **merge** también puede tener una cláusula **when not matched then,** que permite la inserción de nuevos registros en la relación. En el ejemplo anterior, cuando no existen departamentos que casen con ninguna tupla *fondos_recibidos*, la acción de inserción podría crear un nuevo registro departamento (con el *edificio* a *null*) usando la siguiente cláusula:

when not matched then
insert values (*F.nombre_dept*, *null*, *F.presupuesto*)

Aunque no resulte tener mucho sentido en este ejemplo,[1] la cláusula **when not matched then** puede ser de gran utilidad en otros casos. Por ejemplo, suponga que la relación local es una copia de una relación maestra, y se reciben nuevos registros insertados y registros actualizados de la relación maestra. La sentencia **merge** puede actualizar los registros que casan (que actualizarán los registros antiguos) e insertar los que no casan (serán los registros nuevos).

No todas las implementaciones de SQL disponen en la actualidad de la sentencia **merge;** consulte su correspondiente manual del sistema para obtener más detalles.

24.1.3. Localización de los cuellos de botella

El rendimiento de la mayor parte de los sistemas (al menos, antes de ajustarlos) suele quedar limitado de entrada por el rendimiento de uno o varios de sus componentes, denominados **cuellos de botella.** Por ejemplo, puede que un programa pase el ochenta por ciento del tiempo en un pequeño bucle ubicado en las profundidades del código y el veinte por ciento restante en el resto del código; ese pequeño bucle es, pues, un cuello de botella. La mejora del rendimiento de un componente que no sea un cuello de botella hace poco por mejorar la velocidad global del sistema; en este ejemplo, la mejora de la velocidad del resto del código no puede conducir a más de un veinte por ciento de mejora global, mientras que la mejora de la velocidad del bucle cuello de botella puede lograr una mejora de casi el ochenta por ciento global, en el mejor de los casos.

Por tanto, al ajustar un sistema, primero hay que intentar descubrir los cuellos de botella y luego eliminarlos mejorando el rendimiento de los componentes que los generan. Cuando se elimina un cuello de botella puede ocurrir que otro componente se transforme en cuello de botella. En los sistemas bien equilibrados ningún componente aislado constituye un cuello de botella. Si el sistema contiene cuellos de botella se infrautilizan los componentes que no forman parte de los cuellos de botella y, quizás, pudieran haberse sustituido por componentes más económicos con menores prestaciones.

En los programas sencillos, el tiempo en que se ejecuta cada zona del código determina el tiempo global de ejecución. Sin embargo, los sistemas de bases de datos son mucho más complejos y pueden modelarse como **sistemas de colas.** Cada transacción necesita varios servicios del sistema de bases de datos, comenzando por la entrada en los procesos del servidor, las lecturas de disco durante la ejecución, los ciclos de la CPU y los bloqueos para el control de concurrencia. Cada uno de estos servicios tiene asociada una cola, y puede que las transacciones pequeñas pasen la mayor parte del tiempo esperando en las colas —especialmente en las colas de E/S de los discos— en lugar de ejecutando código. La Figura 24.2 muestra algunas de las colas de los sistemas de bases de datos.

Como consecuencia de las numerosas colas de la base de datos, los cuellos de botella de los sistemas de bases de datos suelen manifestarse en forma de largas colas para un servicio determinado o, de modo equivalente, en elevados índices de uso de un servicio concreto. Si las solicitudes se espacian de manera exactamente uniforme, y el tiempo para atender una solicitud es menor o igual que el tiempo antes de que llegue la siguiente solicitud, cada solicitud hallará el recurso sin usar y podrá iniciar la ejecución de manera inmediata, sin esperar. Por desgracia, la llegada de las solicitudes en los sistemas de bases de datos nunca es tan uniforme, sino más bien aleatoria.

Si un recurso (como puede ser un disco) tiene un bajo índice de uso, es probable que este recurso no se esté utilizando al realizar una solicitud, en cuyo caso el tiempo de espera de la solicitud será cero. Suponiendo llegadas distribuidas de forma aleatoria uniforme, la longitud de la cola (y, en consecuencia, el tiempo de espera) aumentará de manera exponencial con el uso; a medida que el uso se aproxime al cien por cien, la longitud de la cola aumentará abruptamente, lo que dará lugar a tiempos de espera excesivamente elevados. El uso de los recursos debe mantenerse lo suficientemente bajo como para que la longitud de la cola sea corta. Como indicativo, los usos cercanos al setenta por ciento se consideran buenos, y los superiores al noventa por ciento se consideran excesivos, dado que generan retrasos significativos. Para aprender más sobre la teoría de los sistemas de colas, generalmente conocida como **teoría de colas,** se pueden consultar las referencias de las notas bibliográficas.

1 Sería preferible haber insertado estos registros en una relación de error, pero esto no se puede hacer con la sentencia **merge.**

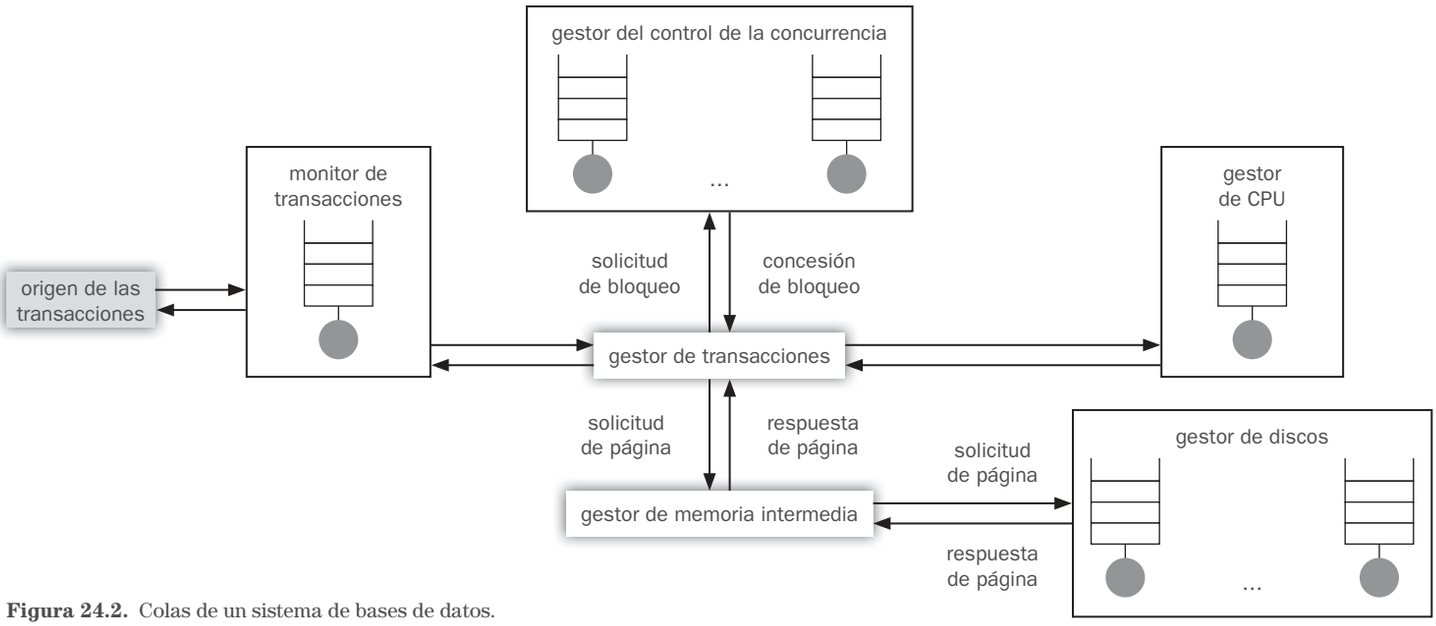


Figura 24.2. Colas de un sistema de bases de datos.

24.1.4. Parámetros ajustables

Los administradores de bases de datos pueden ajustar los sistemas de bases de datos en tres niveles. El nivel inferior es el nivel de hardware. Las opciones para el ajuste de los sistemas en este nivel incluyen añadir discos o usar sistemas RAID si la E/S de disco constituye un cuello de botella, añadir más memoria si el tamaño de la memoria intermedia del disco constituye un cuello de botella o aumentar la velocidad del procesador si el uso de la CPU constituye un cuello de botella.

El segundo nivel consiste en los parámetros de los sistemas de bases de datos, como el tamaño de la memoria intermedia y los intervalos de puntos de revisión. El conjunto exacto de los parámetros de los sistemas de bases de datos que pueden ajustarse depende de cada sistema concreto de bases de datos. Los manuales de los sistemas de bases de datos suelen proporcionar información sobre los parámetros del sistema que se pueden ajustar y sobre el modo en que se deben escoger los valores de esos parámetros. Los sistemas de bases de datos bien diseñados llevan a cabo automáticamente todos los ajustes posibles, lo que libera al usuario o al administrador de la base de datos de esa carga. Por ejemplo, en muchos sistemas de bases de datos el tamaño de la memoria intermedia es fijo pero ajustable. Si el sistema ajusta de manera automática el tamaño de la memoria intermedia observando los indicadores como las tasas de fallo de las páginas, el usuario no tendrá que preocuparse por el ajuste del tamaño de la memoria intermedia.

El tercer nivel es el nivel superior. Incluye el esquema y las transacciones. El administrador puede ajustar el diseño del esquema, los índices que se crean y las transacciones que se ejecutan para mejorar el rendimiento. El ajuste en este nivel es, comparativamente, independiente del sistema.

Los tres niveles de ajuste interactúan entre sí; al ajustar los sistemas hay que considerarlos en conjunto. Por ejemplo, el ajuste en un nivel superior puede hacer que el cuello de botella pase del sistema de discos a la CPU o viceversa.

24.1.5. Ajuste del hardware

Incluso en un sistema de procesamiento de transacciones bien diseñado, cada transacción suele tener que realizar al menos unas cuantas operaciones de E/S si los datos necesarios para la transacción se encuentran en el disco. Un factor importante en el ajuste de un sistema de procesamiento de transacciones es asegurarse de que el subsistema de disco puede admitir la velocidad a la que se solicitan las operaciones de E/S. Por ejemplo, suponga un disco con un tiempo de acceso de unos diez milisegundos y una velocidad media de transferencia de 25 megabytes por segundo (un disco habitual en la actualidad). Este disco soportaría un poco menos de cien operaciones E/S de acceso aleatorio de cuatro kilobytes cada segundo. Si cada transacción necesita exactamente dos operaciones E/S, cada disco soportará, como mucho, cincuenta transacciones por segundo. La única manera de soportar más transacciones por segundo es aumentar el número de discos. Si el sistema tiene que soportar n transacciones por segundo y cada una de ellas lleva a cabo dos operaciones de E/S, hay que dividir (o fragmentar de otra manera) los datos al menos entre $n/50$ discos (ignorando el sesgo).

Hay que tener en cuenta que el factor limitador no es la capacidad del disco, sino la velocidad a la que se puede tener acceso a los datos aleatorios (limitada, a su vez, por la velocidad a la que se puede desplazar el brazo del disco). El número de operaciones de E/S por transacción puede reducirse almacenando más datos en la memoria. Si todos los datos se encuentran en la memoria no habrá operaciones de E/S en el disco salvo por las operaciones de escritura. Guardar los datos usados en la memoria con frecuencia reduce el número de operaciones de E/S y compensa el coste extra de la memoria. Guardar en la memoria los datos que se usan con muy poca frecuencia sería un despilfarro, dado que la memoria es mucho más cara que los discos.

La pregunta es, para una cantidad dada de dinero disponible para gastarlo en discos o en memoria, cuál es la mejor manera de gastar el dinero para obtener el número máximo de transacciones por segundo. Una reducción de una operación de E/S por segundo permite ahorrar:

$$(\textit{precio por unidad de disco})/(\textit{accesos por segundo por disco})$$

Por tanto, si se tiene acceso a una página concreta n veces por segundo, el ahorro debido a guardarla en la memoria es n veces el valor calculado anteriormente. Guardar una página en la memoria cuesta:

$$(\textit{precio por megabyte de memoria})/(\textit{páginas por megabyte de memoria})$$

Así que, el punto de equilibrio es:

$$n * \frac{\textit{precio unidad de disco}}{\textit{accesos por segundo disco}} = \frac{\textit{precio por megabyte de memoria}}{\textit{páginas por megabyte de memoria}}$$

Se puede reordenar la ecuación y sustituir los valores reales por cada uno de los parámetros citados anteriormente para obtener un valor de n; si se tiene acceso a una página con una frecuencia superior a esta, merece la pena comprar suficiente memoria como para almacenarla. La tecnología de discos y los precios actuales de los discos y la memoria (que se supondrán de alrededor de 50 € por disco y 0,02 € por megabyte) dan un valor de n de alrededor de 1/6.400 veces por segundo (o, de manera equivalente, una vez cada dos horas) para las páginas a las que se tiene acceso de manera aleatoria; con el coste de los discos y la memoria de hace algunos años, este mismo valor era de unos cinco minutos.

Este razonamiento se refleja en la recomendación denominada originalmente **regla de los cinco minutos:** si una página se usa más de una vez cada cinco minutos, se debería guardar en la caché de memoria. En otras palabras, hace unos años esta regla sugería comprar suficiente memoria para guardar en la caché todas las páginas a las que se tiene acceso al menos una vez cada cinco minutos de promedio. En la actualidad, vale la pena comprar memoria para hacer caché de todas las páginas a las que se accede al menos una vez cada dos horas, en promedio. Para los datos a los que se tiene acceso con menos frecuencia hay que comprar discos suficientes como para soportar la tasa de E/S exigida por los datos.

Para los datos con acceso secuencial, se puede leer un número de páginas por segundo significativamente mayor. Suponiendo que se lee cada vez un megabyte de datos, se obtiene la **regla del minuto,** que dice que los datos con acceso secuencial deben guardarse en la caché de memoria si se usan al menos una vez por minuto. El número correspondiente, con los costes de memoria y disco actuales de nuestro ejemplo anterior, es de alrededor de 30 segundos. Sorprendentemente, este número no ha cambiado tanto con los años, ya que las tasas de transferencia se han incrementado de manera significativa aunque el precio por megabyte se haya reducido de forma importante, comparado con el precio de un disco.

Claramente, la cantidad de datos que se pueden leer en una operación de E/S afecta de manera significativa al tiempo anterior; de hecho, la regla de los cinco minutos sigue manteniéndose si se leen o escriben en cada operación de E/S unos 100 kilobytes.

La regla de los cinco minutos y sus variantes solo tienen en cuenta el número de operaciones de E/S y no tienen en consideración factores como el tiempo de respuesta. Algunas aplicaciones necesitan guardar en la memoria incluso los datos que se usan con poca frecuencia para soportar tiempos de respuesta inferiores o similares al tiempo de acceso al disco.

Con la gran disponibilidad de memorias flash y «discos de estado sólido» basados en memoria flash, los diseñadores de sistemas puede elegir ahora almacenar los datos de uso frecuente en almacenamiento flash, en lugar de guardarlos en disco. Alternativamente, en el enfoque de **memoria intermedia flash** se utiliza almacenamiento flash como memoria intermedia persistente, asignando a cada bloque una ubicación permanente en el disco, pero guardado en flash en lugar de escribirse en el disco mientras se utilice con frecuencia. Cuando el almacenamiento flash está lleno, se expulsa un bloque de los menos utilizados y se vuelca al disco si tuvo alguna actualización desde la escritura anterior.

El enfoque de memoria intermedia flash requiere cambios en el propio sistema de bases de datos. Aunque el sistema no lo soporte, el administrador puede controlar la asociación de relaciones o índices a discos y asignar los índices o relaciones de uso más frecuente al almacenamiento flash. La función de espacio de tablas, de que disponen la mayoría de sistemas de bases de datos, se puede utilizar para controlar la asociación, creando un espacio de tablas en el almacenamiento flash y asignando las relaciones e índices deseados a dicho espacio de tablas. El control de esta asociación con una granularidad más fina requiere modificar el código del sistema de bases de datos.

La regla de los cinco minutos se ha extendido al caso en que los datos se almacenen en memoria flash, además de en memoria principal y en disco. Consulte las notas bibliográficas para más información.

Otro aspecto del ajuste es si se debe usar RAID 1 o RAID 5. La respuesta depende de la frecuencia con que se actualicen los datos, dado que RAID 5 es mucho más lento que RAID 1 en las operaciones de escritura aleatoria: RAID 5 necesita dos operaciones de lectura y dos de escritura para ejecutar una sola solicitud de escritura aleatoria. Si una aplicación realiza r operaciones de lectura aleatoria y e operaciones de escritura aleatoria por segundo para soportar un intercambio concreto, una implementación de RAID 5 necesitaría $r + 4e$ operaciones de E/S por segundo, mientras que una implementación de RAID 1 necesitaría $r + e$ operaciones de E/S por segundo. Se puede calcular el número de discos necesario para soportar las operaciones de E/S requeridas por segundo dividiendo el resultado del cálculo por cien operaciones de E/S por segundo (para los discos de la generación actual). Para muchas aplicaciones, r y e son lo bastante grandes como para que $(r + e)/100$ discos puedan guardar con facilidad dos copias de todos los datos. Para esas aplicaciones, si se usa RAID 1, ¡el número necesario de discos es realmente menor que el número necesario de discos si se usa RAID 5! Por tanto, RAID 5 solo resulta útil cuando los requisitos de almacenamiento de datos son muy grandes pero las velocidades de E/S y los requisitos de transferencia de datos son pequeños, es decir, para datos muy grandes y muy «fríos».

24.1.6. Ajuste del esquema

Dentro de las restricciones de la forma normal escogida, es posible dividir las relaciones verticalmente. Por ejemplo, considere la relación *asignatura*, con el esquema:

asignatura(*asignatura_id*, *nombre_asig*, *nombre_dept*, *créditos*)

para la que *asignatura_id* es una clave. Dentro de las restricciones de las formas normales (formas normales FNBC y tercera), se puede dividir la relación *asignatura* en dos relaciones:

asignatura_créditos(*asignatura_id*, *créditos*)
asignatura_nombre_dept (*asignatura_id*, *nombre_asig*, *nombre_dept*)

Las dos representaciones son lógicamente equivalentes, dado que *asignatura_id* es una clave, pero tienen características de rendimiento diferentes.

Si la mayor parte de los accesos a la información de la asignatura solo examinan *asignatura_id* y *créditos*, pueden ejecutarse sobre la relación *asignatura_créditos*, y es probable que el acceso resulte algo más rápido, dado que no se obtienen los atributos *nombre_asig* y *nombre_dept*. Por el mismo motivo, cabrán en la memoria intermedia más tuplas de *asignatura_créditos* que las correspondientes tuplas de *asignatura*, lo que vuelve a generar un mayor rendimiento. Este efecto sería especialmente destacado si los atributos *nombre_asig* y *nombre_dept* fueran de gran tamaño. Por tanto, en este caso un esquema que consistiera en *asignatura_créditos* y *asignatura_nombre_dept* sería preferible a otro que consistiera en la relación *asignatura*.

Por otro lado, si la mayor parte de los accesos a la información de la asignatura necesitan tanto *nombre_dept* como *créditos*, será preferible el uso de la relación *asignatura*, dado que se evitará el coste de la reunión de *asignatura_créditos* y *asignatura_nombre_dept*. Además, la sobrecarga de almacenamiento sería menor, dado que solo habría una relación y no se replicaría el atributo *asignatura_id*.

El enfoque de **almacenamiento en columna** para guardar la información se basa en una división vertical, pero lo lleva al límite almacenando cada atributo (o columna) de la relación en un archivo distinto. El almacenamiento en columna se ha demostrado que tiene un buen rendimiento para distintas aplicaciones de almacenes de datos.

Otro truco para mejorar el rendimiento es guardar una *relación desnormalizada*, como puede ser una reunión de *profesor* y de *departamento*, en la que la información de *nombre_dept*, *edificio* y *presupuesto* se repitiera para cada profesor. Hay que realizar un mayor esfuerzo para asegurarse de que la relación es consistente siempre que se realice una actualización. No obstante, una consulta que obtenga los nombres de los profesores y los edificios asociados se aceleraría, dado que se habría calculado previamente la reunión de *profesor* con *departamento*. Si se ejecuta con frecuencia una consulta de este tipo, y hay que llevarla a cabo con la máxima eficiencia posible, puede resultar beneficiosa la relación desnormalizada.

Las vistas materializadas pueden proporcionar las ventajas que ofrecen las relaciones desnormalizadas, al coste de algún almacenamiento extra; el ajuste del rendimiento de las vistas materializadas se describe en la Sección 24.1.8. Una de las principales ventajas de las vistas materializadas respecto de las relaciones desnormalizadas es que el mantenimiento de la consistencia de los datos redundantes pasa a ser labor del sistema de bases de datos, no del programador. Por tanto, las vistas materializadas resultan preferibles, siempre que las soporte el sistema de bases de datos.

Otro enfoque para acelerar el cálculo de la reunión sin materializarla es agrupar los registros que coincidirán en la reunión en la misma página del disco. Estas organizaciones agrupadas de archivos se vieron en la Sección 10.6.2.

24.1.7. Ajuste de los índices

Los índices de un sistema de bases de datos se pueden ajustar para mejorar el rendimiento. Si las consultas constituyen el cuello de botella, se las suele poder acelerar creando los índices adecuados en las relaciones. Si lo constituyen las actualizaciones, puede que haya demasiados índices, que hay que actualizar cuando se actualizan las relaciones. La eliminación de índices puede que acelere algunas actualizaciones.

La elección del tipo de índice también es importante. Algunos sistemas de bases de datos soportan diferentes tipos de índices, como los índices asociativos y los índices de árboles B. Si las consultas más frecuentes son de intervalo, es preferible usar índices de árboles B que índices asociativos. Otro parámetro ajustable es la posibilidad de hacer que un índice tenga agrupación. Solo se puede hacer un índice con agrupación por relación, guardando la relación ordenada por los atributos del índice. Generalmente conviene hacer el índice con agrupación que beneficie al mayor número de consultas y de actualizaciones.

Para ayudar a identificar los índices que se deben crear y el índice (si es que hay alguno) de cada relación que se debe agrupar, la mayoría de los sistemas de bases de datos comerciales proporcionan *asistentes para el ajuste*; se describen con más detalle en la Sección 24.1.9. Estas herramientas usan el historial de consultas y de actualizaciones (denominado *carga de trabajo*) para estimar los efectos de varios índices en el tiempo de ejecución de las consultas y de las actualizaciones en la carga de trabajo. Las recomendaciones sobre los índices que se deben crear se basan en estas estimaciones.

24.1.8. Uso de vistas materializadas

El uso de las vistas materializadas puede acelerar enormemente ciertos tipos de consultas, en especial las consultas de agregación. Recuérdese el ejemplo de la Sección 13.5, en el que se solicitaba con frecuencia el sueldo total de los departamentos (obtenido sumando los sueldos de todos los profesores del departamento). Como se vio en esa sección, la creación de una vista materializada que guarde el sueldo total de los departamentos puede acelerar enormemente estas consultas.

Las vistas materializadas, no obstante, deben usarse con cuidado, dado que almacenarlas no solo supone una sobrecarga de espacio sino que, lo que es más importante, su mantenimiento también supone una sobrecarga de tiempo. En el caso del **mantenimiento inmediato de las vistas,** si las actualizaciones de una transacción afectan a la vista materializada hay que actualizarla como parte de la misma transacción. Por tanto, puede que la transacción se ejecute más lentamente. En el caso del **mantenimiento diferido de las vistas,** la vista materializada se actualiza posteriormente; hasta que se actualice puede que la vista materializada sea inconsistente con las relaciones de la base de datos. Por ejemplo, puede que la vista materializada se actualice cuando la utilice una consulta o que se actualice de manera periódica. El uso del mantenimiento diferido reduce la carga de las transacciones de actualización.

El administrador del sistema es el responsable de la selección de las vistas materializadas y las directivas de mantenimiento. El administrador del sistema puede realizar la selección de modo manual examinando los tipos de consultas de la carga de trabajo y averiguando las consultas que necesitan ejecutarse más rápidamente y las actualizaciones o consultas que pueden ejecutarse más lentamente. A partir del examen, el administrador del sistema puede escoger un conjunto adecuado de vistas materializadas. Por ejemplo, puede que el administrador descubra que se usa con frecuencia un agregado determinado y decida materializarlo, o puede que observe que una reunión concreta se calcula con frecuencia y decida materializarla.

Sin embargo, la selección manual resulta tediosa incluso para conjuntos de tipos de consultas moderadamente grandes y puede que resulte difícil realizar una buena selección, dado que exige comprender los costes de diferentes alternativas; solo el optimizador de consultas puede estimar los costes con una precisión razonable, sin ejecutar realmente la consulta. Por tanto, es posible que solo se halle un buen conjunto de vistas mediante el procedimiento de prueba y error; es decir, materializando una o varias vistas, ejecutando la carga de trabajo y midiendo el tiempo utilizado para ejecutar las consultas de la carga de trabajo. El administrador repetirá el proceso hasta que encuentre un conjunto de vistas que ofrezca un rendimiento aceptable.

Una opción mejor es proporcionar soporte para la selección de las vistas materializadas desde el interior del propio sistema de bases de datos, integrado con el optimizador de consultas. Este enfoque se describe con más detalle en la Sección 24.1.9.

24.1.9. Ajuste automático del diseño físico

La mayoría de los sistemas comerciales de bases de datos actuales ofrecen herramientas para ayudar al administrador de la base de datos en la selección de los índices y de las vistas materializadas, así como para otras tareas relacionadas con el diseño de bases de datos tales como la división de datos en sistemas de bases de datos paralelos.

Estas herramientas examinan la **carga de trabajo** (el historial de consultas y de actualizaciones) y sugieren los índices y las vistas que hay que materializar. El usuario de la herramienta de ajuste tiene la posibilidad de especificar la importancia de acelerar las diferentes consultas, lo que el usuario tiene en cuenta al seleccionar las vistas que vaya a materializar. A menudo el ajuste se debe hacer antes de que se desarrolle completamente la aplicación, y el contenido real de la base de datos puede ser pequeño para la de desarrollo, pero se espera que sea mucho más grande en la de producción. Así, algunas herramientas de ajuste también permiten que el usuario especifique la información sobre el tamaño previsto de la base de datos y de las estadísticas relacionadas.

Database Tuning Assistant de Microsoft, por ejemplo, permite que el usuario formule preguntas del tipo «¿qué pasaría si…?», por lo que el usuario puede escoger una vista y el optimizador estimará el efecto de materializarla en el coste total de la carga de trabajo y en los costes individuales de los diferentes tipos de consultas o de actualizaciones en la carga de trabajo.

Las técnicas de selección automática de índices y vistas materializadas generalmente se implementan enumerando varias alternativas y usando el optimizador de consultas para estimar los costes y las ventajas de seleccionar cada alternativa usando la carga de trabajo. Puesto que el número de las alternativas de diseño puede ser extremadamente grande, así como la carga de trabajo, las técnicas de selección se deben diseñar cuidadosamente.

El primer paso es generar una carga de trabajo. Esto se hace generalmente registrando todas las consultas y actualizaciones que se ejecutan durante un cierto periodo. Después, las herramientas de selección realizan una **compresión de la carga de trabajo,** es decir, crean una representación de la carga de trabajo usando un número pequeño de actualizaciones y de consultas. Por ejemplo, las actualizaciones con la misma forma se pueden representar con una sola actualización con un peso que corresponde a cuántas veces se produjo la actualización. Las consultas con la misma forma se pueden sustituir análogamente por un representante con el peso apropiado. Después de esto, las preguntas que son muy infrecuentes y no tienen un alto coste se pueden desechar del análisis. Las preguntas más costosas se pueden elegir para tratarlas en primer lugar. La compresión de la carga de trabajo es esencial para grandes cargas de trabajo.

Con la ayuda del optimizador, la herramienta calcularía un conjunto de índices y de vistas materializadas que mejorarían las consultas y actualizaciones en la compresión de la carga de trabajo. Se pueden probar diversas combinaciones de estos índices y vistas materializadas para encontrar la mejor combinación. Sin embargo, un enfoque exhaustivo sería impracticable, puesto que el potencial número de índices y vistas materializadas es ya grande, y cada

subconjunto de estos es una alternativa potencial de diseño, conduciendo a un número exponencial de alternativas. Se usan heurísticas para la reducción del espacio de alternativas, es decir, para reducir el número de las combinaciones consideradas.

Las heurísticas ávidas de la selección de índices y vistas materializadas funcionan como se explica a continuación. Estiman las ventajas de la materialización de las diferentes vistas e índices (usando la estimación del coste del optimizador). Se escoge la vista o índice que ofrece la máxima ganancia o la máxima ventaja por unidad de espacio (es decir, la ventaja dividida por el espacio necesario para guardar la vista o el índice). Se debe tomar en consideración al calcular la ganancia el coste del mantenimiento de la vista o el índice. Una vez que la heurística ha seleccionado una vista o índice puede que haya cambiado la ganancia de otras vistas o índices, por lo que la heurística vuelve a calcularlas y escoge la segunda mejor vista para su materialización. El proceso continúa hasta que se agota el espacio de disco disponible para guardar las vistas materializadas e índices, o bien cuando el coste de mantenimiento del resto de candidatos es superior a la ganancia de las consultas que los usen.

Las herramientas del mundo real para la selección de índices y vistas materializadas generalmente incorporan algunos elementos de la selección ávida, pero usan otras técnicas para conseguir mejores resultados. También contemplan otros aspectos del diseño físico de bases de datos, tales como decidir la forma de dividir las relaciones en las bases de datos paralelas, o el mecanismo físico de almacenaje a usar para una relación.

24.1.10. Ajuste de las transacciones concurrentes

La ejecución concurrente de diferentes tipos de transacciones puede generar en algunos casos un bajo rendimiento debido a la disputa por bloqueos. En primer lugar se considerará la disputa de lectura-escritura, que es la más común, y después se considera la disputa de escritura-escritura.

Como ejemplo de **disputa de lectura-escritura,** considere la siguiente situación de una base de datos de un banco. Durante el día, se realizan numerosas actualizaciones casi continuamente. Suponga que en ese tiempo se lleva a cabo una gran consulta de cálculo de estadísticas de las distintas sucursales. Si la consulta realiza el examen de una relación, puede bloquear todas las actualizaciones de esa relación mientras se ejecuta, y puede tener un efecto desastroso en el rendimiento del sistema.

Muchos sistemas de bases de datos (Oracle, PostgreSQL y SQL Server de Microsoft, por ejemplo) disponen del aislamiento de instantáneas, en el que las consultas se ejecutan en una instantánea de los datos y las actualizaciones se realizan concurrentemente. (El aislamiento de instantáneas se describe con detalle en la Sección 15.7). El aislamiento de instantáneas se debería utilizar, si se dispone del mismo, en grandes consultas, para evitar la disputa de bloqueos de la situación anterior. En SQL Server, cuando se ejecuta la sentencia

set transaction isolation level snapshot

al comienzo de una transacción, se genera una instantánea que se utiliza para dicha transacción. En Oracle y PostgreSQL, usando la palabra clave **serializable** en lugar de **snapshot** en el comando anterior se produce el mismo efecto, ya que ambos sistemas utilizan realmente el aislamiento de instantáneas cuando el nivel de aislamiento se establece a serializable.

Si no se dispone de aislamiento de instantáneas, una opción alternativa es ejecutar las consultas grandes en aquellos momentos en que haya pocas actualizaciones. Sin embargo, para las bases de datos que ofrecen servicios web, puede que estos periodos tranquilos para las actualizaciones no existan.

Otra alternativa es utilizar niveles de consistencia más débiles, como el nivel de aislamiento **lectura comprometida,** en el que la evaluación de la consulta tenga un impacto mínimo en las actualizaciones concurrentes, pero no se garantiza que el resultado de la consulta sea consistente. La semántica de la aplicación es la que determina si esta respuesta aproximada (inconsistente) es aceptable.

Vamos a tratar el caso de la **disputa de escritura-escritura.** Los datos que se actualizan con mucha frecuencia pueden generar un pobre rendimiento con los bloqueos, ya que muchas transacciones pueden estar esperando por bloqueos sobre esos datos. Estos datos se denominan **puntos calientes de actualización.** Estos puntos calientes pueden generar problemas incluso con aislamiento de instantáneas, lo que genera con frecuencia que las transacciones aborten debido al fallo de la validación de escritura. Una situación común que se genera en un punto caliente de actualización es la siguiente: las transacciones necesitan asignar identificadores únicos a los datos que se insertan en la base de datos, por lo que leen e incrementan un contador secuencial almacenado en una tupla de la base de datos. Si las inserciones no son frecuentes, y el contador secuencial se bloquea en dos fases, la tupla que contiene el contador secuencial se convierte en un punto caliente.

Una forma de mejorar la concurrencia es liberar el bloque sobre el contador secuencial, en cuanto se lee e incrementa; sin embargo, al hacerlo de esta forma, si la transacción aborta, la actualización del contador secuencial debe volver atrás. Para entender por qué, suponga que T_1 incrementa el contador secuencial, después T_2 incrementa el contador secuencial antes de que T_1 comprometa; si T_1 aborta, al volver atrás la actualización, ya sea restaurando el contador al valor original o decrementando el contador, generará el valor de secuencia que usa T_2, que se reutilizará en la subsiguiente transacción.

La mayoría de las bases de datos proporcionan un constructor especial para crear **contadores secuenciales** que implementan la liberación de bloqueo temprana, no de dos fases, con un tratamiento especial de la operación deshacer, por lo que las actualizaciones del contador no se vuelven atrás si la transacción aborta. La norma de SQL permite crear un contador secuencial con el comando:

create sequence *contador1*;

En el comando anterior, *contador1* es el nombre de la secuencia; se pueden crear secuencias con distintos nombres. La sintaxis para obtener un valor de una secuencia no está normalizada. En Oracle, *contador1.nextval* devuelve el siguiente valor de la secuencia, tras incrementarla, y este mismo efecto tendría la llamada a la función *nextval*(´contador1´) en PostgreSQL, y en DB2 se usa la sintaxis **nextval for** *contador1*.

La norma de SQL proporciona alternativas útiles al uso de un contador secuencial explícito cuando el objetivo es proporcionar identificadores únicos a las tuplas que se insertan en una relación. Para ello, se añade la palabra clave **identity** a la declaración de un atributo entero de una relación (normalmente este atributos se debería declarar también como clave primaria). Si se deja sin especificar un valor para este atributo en una sentencia de inserción, se crea un nuevo valor único automáticamente con cada nueva tupla insertada. Se usa internamente un contador secuencial que no usa bloqueo de dos fases para implementar la declaración **identity,** incrementando el contador cada vez que se inserta una tupla. Hay varias bases de datos que disponen de la declaración **identity,** como DB2 o SQL Server, aunque la sintaxis varía. PostgreSQL dispone de un tipo de dato llamado **serial,** que proporciona el mismo efecto; el tipo **serial** de PostgreSQL se implementa de forma transparente creando una secuencia con un bloqueo que no es de dos fases.

Es importante destacar que aunque la adquisición de un número de secuencia por una transacción no se puede volver atrás si la transacción aborta (por razones ya tratadas), al abortar la tran-

sacción se puede generar un *salto en el número de secuencia* en las tuplas insertadas en la base de datos. Por ejemplo, puede haber tuplas con los identificadores 1001 y 1003 y no con el número 1002, si la transacción que consiguió el número 1002 no pasó al estado comprometida. Estos saltos no son aceptables para algunas aplicaciones, por ejemplo, en aplicaciones financieras en las que no puede haber saltos en los números de recibos o de facturas. Para estas aplicaciones no se pueden usar las secuencias que proporcionan las bases de datos con atributos que se incrementan automáticamente, ya que se generan huecos. Un contador secuencial que se almacene en una tupla normal y con un bloqueo de dos fases no sufrirá estos saltos, ya que si una transacción aborta se restaurará el valor del contador y la siguiente transacción seguiría con la misma secuencia evitando los saltos.

Las transacciones con actualización prolongada pueden generar problemas de rendimiento en los registros del sistema e incrementar el tiempo que le lleva al sistema recuperarse si se produce un fallo. Si una transacción realiza muchas actualizaciones, el registro del sistema se puede llenar incluso antes de que la transacción se complete, en cuyo caso hay que volver atrás. Si una transacción de actualización se ejecuta durante un periodo muy largo (incluso con pocas actualizaciones), puede bloquear el borrado de partes antiguas del registro, si el sistema de registro histórico no está muy bien diseñado. De nuevo, este bloqueo puede hacer que el registro se acabe llenando.

Para evitar estos problemas, muchos sistemas de bases de datos imponen límites estrictos en el número de actualizaciones que puede realizar una única transacción. Aunque el sistema no imponga ningún límite, suele ser conveniente dividir una transacción de actualización larga en un conjunto más pequeño de transacciones de actualización, si esto es posible. Por ejemplo, una transacción que actualiza todos los sueldos de los empleados de una gran corporación incrementándolos podría dividirla en un conjunto de pequeñas transacciones, donde cada una actualiza un conjunto reducido de empleados. Estas transacciones se denominan **transacciones en minilotes.** Sin embargo, las transacciones en minilotes se deben realizar con cuidado. En primer lugar, si existen actualizaciones concurrentes sobre el conjunto de empleados, puede que el resultado del conjunto de pequeñas transacciones no sea equivalente al de la transacción grande. En segundo lugar, si se produce un fallo se incrementan los salarios de algunos de los trabajadores por transacciones comprometidas, mientras los de otros no. Para evitar este problema, en cuanto el sistema se recupere del fallo hay que ejecutar las transacciones que permanezcan en el lote.

Las transacciones prolongadas, ya sean de solo lectura o de actualización, pueden generar que la tabla de bloqueos se llene. Si una única consulta explora una gran relación, el optimizador de consultas debería asegurar que se obtiene un bloqueo de la relación en lugar de un gran número de bloqueos de tupla. Sin embargo, si una transacción ejecuta un gran número de pequeñas consultas o actualizaciones, puede necesitar obtener un gran número de bloqueos, los que genera que la tabla de bloqueos se pueda llegar a llenar.

Para evitar este problema, algunas bases de datos proporcionan una **escalada de bloqueos** automática; con esta técnica, si una transacción ha obtenido un gran número de bloqueos sobre tuplas, los bloqueos de tupla se actualizan a bloqueos de página, o incluso a bloqueos de toda la relación. Recuerde que con el bloqueo de múltiple granularidad (Sección 15.3), cuando se obtiene un bloqueo de mayor nivel no son necesarios los bloqueos de nivel más fino, por lo que las entradas de bloqueo de tupla se pueden eliminar de la tabla de bloqueos, liberando espacio. En las bases de datos que no disponen de escalada de bloqueos es posible que la propia transacción adquiera explícitamente un bloqueo sobre una relación, evitando por tanto la obtención de bloqueos de tupla.

24.1.11. Simulación de rendimiento

Para comprobar el rendimiento de un sistema de bases de datos incluso antes de instalarlo se puede crear un modelo de simulación del rendimiento de ese sistema. En la simulación se modela cada servicio que aparece en la Figura 24.2, como la CPU, cada disco, la memoria intermedia y el control de concurrencia. En lugar de modelar los detalles de un servicio, puede que el modelo de simulación solo capture algunos aspectos de cada uno, como el **tiempo de servicio;** es decir, el tiempo que tarda en acabar de procesar una solicitud una vez comenzado el procesamiento. Por tanto, la simulación puede modelar el acceso a disco partiendo solo del tiempo medio de acceso.

Dado que las solicitudes de un servicio suelen tener que esperar su turno, en el modelo de simulación cada servicio tiene asociada una cola. Cada transacción consiste en una serie de solicitudes. Las solicitudes se disponen en una cola según llegan y se atienden de acuerdo con la política de cada servicio, como puede ser que el primero en llegar sea el primero en ser atendido. Los modelos de los servicios como la CPU y los discos operan conceptualmente en paralelo, para tener en cuenta el hecho de que estos subsistemas operan en paralelo en los sistemas reales.

Una vez creado el modelo de simulación para el procesamiento de las transacciones, el administrador del sistema puede ejecutar en él varios experimentos. El administrador puede usar los experimentos con transacciones simuladas que lleguen con diferentes velocidades para averiguar el modo en que se comportaría el sistema bajo diferentes condiciones de carga. También puede ejecutar otros experimentos que varíen los tiempos de servicio de cada servicio para averiguar la sensibilidad del rendimiento a cada uno de ellos. También se pueden variar los parámetros del sistema de modo que se pueda realizar el ajuste del rendimiento en el modelo de simulación.

24.2. Pruebas de rendimiento

A medida que se van estandarizando los servidores de bases de datos, el factor diferenciador entre los productos de los diversos fabricantes es el rendimiento de estos. Las **pruebas de rendimiento** son conjuntos de tareas que se usan para cuantificar el rendimiento de los sistemas de software.

24.2.1. Familias de tareas

Dado que la mayor parte de los sistemas de software, como las bases de datos, son complejos, existe una gran variación en su implementación por parte de los diferentes fabricantes. En consecuencia, hay una variación significativa en su rendimiento en las diferentes tareas. Puede que un sistema sea el más eficiente en una tarea concreta y puede que otro lo sea en una tarea diferente. Por tanto, una sola tarea no suele resultar suficiente para cuantificar el rendimiento del sistema. Por ello, el rendimiento de un sistema se mide mediante familias de tareas estandarizadas, denominadas *pruebas de rendimiento*.

La combinación de los resultados de rendimiento de varias tareas debe realizarse con cuidado. Suponga que se tienen dos tareas, T_1 y T_2, y que se mide la productividad de un sistema como el número de transacciones de cada tipo que se ejecutan en un tiempo dado (digamos, un segundo). Suponga que el sistema A ejecuta T_1 a noventa y nueve transacciones por segundo y que T_2 se ejecuta a una transacción por segundo. De manera parecida, suponga que el sistema B ejecuta T_1 y T_2 a cincuenta transacciones por segundo. Suponga también que una carga de trabajo tiene una mezcla a partes iguales de los dos tipos de transacciones.

Si se toma el promedio de los dos pares de resultados (es decir, 99 y 1 frente a 50 y 50), pudiera parecer que los dos sistemas tienen el mismo rendimiento. Sin embargo, sería *erróneo* tomar los promedios de esta manera; si se ejecutaran cincuenta transacciones de cada tipo el sistema A tardaría unos 50.5 segundos en concluirlas, mientras que el sistema B las terminaría ¡en solo 2 segundos!

Este ejemplo muestra que una sola medida del rendimiento induce a error si hay más de un tipo de transacción. El modo correcto de promediar los números es tomar el **tiempo para concluir** la carga de trabajo, en vez del rendimiento promedio de cada tipo de transacción. De esta forma se puede calcular con exactitud el rendimiento del sistema en transacciones por segundo para una carga de trabajo concreta. Por tanto, el sistema A tarda 50.5/100, que son 0.505 segundos por transacción, mientras que el sistema B tarda 0.02 segundos por transacción, en promedio. En términos de productividad, el sistema A se ejecuta a un promedio de 1.98 transacciones por segundo, mientras que el sistema B se ejecuta a 50 transacciones por segundo. Suponiendo que las transacciones de todos los tipos son igual de probables, el modo correcto de promediar la productividad respecto de los diferentes tipos de transacciones es tomar la **media armónica** de las productividades. La media armónica de n productividades $t_1, \ldots t_n$ se define como:

$$\frac{n}{\frac{1}{t_1}+\frac{1}{t_2}+\ldots+\frac{1}{t_n}}$$

Para este ejemplo, la media armónica del rendimiento en el sistema A es 1.98. Para el sistema B es 50. Por tanto, el sistema B es aproximadamente veinticinco veces más rápido que el sistema A para una carga de trabajo consistente en una mezcla a partes iguales de los dos tipos de transacciones de ejemplo.

24.2.2. Clases de aplicaciones de bases de datos

El **procesamiento de transacciones en línea** (*online transaction processing*: OLTP) y la **ayuda a la toma de decisiones** (incluyendo el **procesamiento analítico en línea** (*online analytical processing*: OLAP) son dos grandes clases de aplicaciones de los sistemas de bases de datos. Estas dos clases de tareas tienen necesidades diferentes. Se necesita una elevada concurrencia y técnicas inteligentes para acelerar el procesamiento de las operaciones de compromiso a la hora de soportar una elevada tasa de transacciones de actualización. Por otro lado, son necesarios buenos algoritmos para la evaluación de consultas y la optimización de las mismas para la ayuda a la toma de decisiones. La arquitectura de algunos sistemas de bases de datos se ha ajustado para el procesamiento de las transacciones; la de otros, como las series de Teradata de sistemas paralelos de bases de datos, para la ayuda a la toma de decisiones. Otros fabricantes intentan conseguir un equilibrio entre las dos tareas.

Las aplicaciones suelen tener una mezcla de necesidades de procesamiento de transacciones y ayuda a la toma de decisiones. Por tanto, el mejor sistema de bases de datos para cada aplicación depende de la mezcla de las dos necesidades que tenga la aplicación.

Suponga que dispone de resultados de rendimiento para las dos clases de aplicaciones por separado y la aplicación en cuestión tiene una mezcla de transacciones de las dos clases. Hay que ser precavido incluso al tomar la media armónica de los resultados de rendimiento, debido a la **interferencia** entre las transacciones. Por ejemplo, una transacción de ayuda a la toma de decisiones que tarde mucho en ejecutarse puede adquirir varios bloqueos, lo que puede evitar el progreso de las transacciones de actualización. La media armónica de las productividades solo debe usarse si las transacciones no interfieren entre sí.

24.2.3. Las pruebas de rendimiento TPC

El **Consejo para el rendimiento del procesamiento de las transacciones** (*Transaction Processing Performance Council*: TPC) ha definido una serie de normas de pruebas de rendimiento para los sistemas de bases de datos.

Las pruebas TPC se definen con gran minuciosidad. Definen el conjunto de relaciones y el tamaño de las tuplas no como un número fijo, sino como un múltiplo del número de transacciones por segundo que se afirma que se realizan, para reflejar que una tasa mayor de ejecución de transacciones probablemente se halle correlacionada con un número mayor de cuentas. La métrica del rendimiento es la productividad, expresado como **transacciones por segundo (TPS).** Cuando se mide el rendimiento, el sistema debe proporcionar un tiempo de respuesta que se halle dentro de ciertos límites, de modo que no pueda obtenerse un rendimiento elevado a expensas de tiempos de respuesta muy elevados. Además, para las aplicaciones profesionales, el coste es de gran importancia. Por tanto, la prueba TPC también mide el rendimiento en términos de **precio por TPS.** Puede que los sistemas de gran tamaño tengan un elevado número de transacciones por segundo, pero puede que resulten caros (es decir, que tengan un precio elevado por TPS). Además, una compañía no puede afirmar que tiene resultados de las pruebas TPC en sus sistemas *sin* una auditoría externa que asegure que el sistema sigue fielmente la definición de la prueba, incluyendo el soporte pleno de las propiedades ACID de las transacciones.

La primera de la serie fue la **prueba TPC-A,** que se definió en 1989. Esta prueba simula una aplicación bancaria típica mediante un solo tipo de transacción que modela la retirada y el depósito de efectivo en un cajero automático. La transacción actualiza varias relaciones —como el saldo del banco, el saldo del cajero y el saldo del cliente— y añade un registro a una relación de seguimiento para la auditoría. La prueba también incorpora la comunicación con los terminales para modelar el rendimiento de extremo a extremo del sistema de manera realista. La **prueba TPC-B** se diseñó para probar el rendimiento central del sistema de bases de datos, junto con el sistema operativo en el que se ejecuta. Elimina las partes de la prueba TPC-A que tratan con los usuarios, las que tratan de las comunicaciones y las que tratan de los terminales para centrarse en el servidor de bases de datos dorsal. Ni TPC-A ni TPC-B se usan mucho hoy en día.

La **prueba TPC-C** se diseñó para modelar un sistema más complejo que el de la prueba TPC-A. La prueba TPC-C se concentra en las actividades principales de un entorno de admisión de pedidos, como son la entrada y la entrega de pedidos, el registro de los pagos, la verificación del estado de los pedidos y el seguimiento de los niveles de inventarios. La prueba TPC-C se sigue usando habitualmente para la prueba de sistemas de procesamiento de transacciones (OLTP). La prueba TPC-E, más reciente, también trata sobre los sistemas OLTP, pero se basa en un modelo de una firma de bolsa, con usuarios que interaccionan con la firma y generar transacciones. La firma por su parte interacciona con los mercados financieros para ejecutar transacciones.

La **prueba TPC-D** se diseñó para probar el rendimiento de los sistemas de bases de datos en consultas de ayuda a la toma de decisiones. Los sistemas de ayuda a la toma de decisiones se están volviendo cada vez más importantes hoy en día. Las pruebas TPC-A, TPC-B y TPC-C miden el rendimiento de las cargas de procesamiento de transacciones y no deben usarse como medida del rendimiento en consultas de ayuda a la toma de decisiones. La D de TPC-D viene de **ayuda a la toma de decisiones.** El esquema de la prueba TPC-D modela una aplicación de ventas/distribución, con artículos, clientes y pedidos, junto con cierta información auxiliar. El tamaño de las relaciones se define como una relación y el tamaño de la base de datos es el tamaño total de todas las relacio-

nes, expresado en gigabytes. TPC-D con un factor de escala de uno representa la prueba TPC-D para una base de datos de un gigabyte, mientras que el factor de escala diez representa una base de datos de diez gigabytes. La carga de trabajo de la prueba consiste en un conjunto de 17 consultas SQL que modelan tareas que se ejecutan frecuentemente en los sistemas de ayuda a la toma de decisiones. Parte de las consultas usan características complejas de SQL, como las consultas de agregación y las consultas anidadas.

Los usuarios de las pruebas pronto se dieron cuenta de que las diferentes consultas TPC-D podían acelerarse de manera significativa usando las vistas materializadas y otra información redundante. Hay aplicaciones, como las tareas periódicas de información, en las que las consultas se conocen con antelación y las vistas materializadas pueden seleccionarse con cuidado para acelerar las consultas. Resulta necesario, no obstante, tener en cuenta la sobrecarga que supone mantener las vistas materializadas.

La **prueba TPC-H** (donde la H viene de **ad hoc**) supone un refinamiento de la prueba TPC-D. El esquema es el mismo, pero hay 22 consultas, de las cuales 16 provienen de TPC-D. Además, hay dos actualizaciones, un conjunto de inserciones y un conjunto de borrados. TPC-H prohíbe vistas materializadas y otra información redundante, y solo permite índices en las claves primaria y externa. Estas pruebas modelan consultas ad hoc en las que las consultas no se conocen de antemano, por lo que no es posible crear las vistas materializadas con antelación. Una variante, TPC-R, donde R viene de «reporting», que ya no se usa, permite el uso de vistas materializadas y otra información redundante.

TPC-H mide el rendimiento de esta forma: la **prueba de potencia** ejecuta las consultas y las actualizaciones una a una de manera secuencial, y 3.600 segundos divididos por la media geométrica de los tiempos de ejecución de las consultas (en segundos) da una medida de las consultas por hora. La **prueba de rendimiento** ejecuta varias secuencias en paralelo, cada una de las cuales ejecuta las 22 consultas. También hay una corriente de actualizaciones paralela. Aquí el tiempo total de toda la ejecución se usa para calcular el número de consultas por hora.

La **métrica compuesta consultas por hora,** que es la métrica global, se obtiene como la raíz cuadrada del producto de las métricas de potencia y de rendimiento. Se define una **métrica compuesta precio/rendimiento** dividiendo el precio del sistema por la métrica compuesta.

La **prueba de comercio web (TPC-W)** es una prueba de extremo a extremo que modela los sitios web que poseen contenido estático (imágenes, sobre todo) y contenido dinámico generado a partir de una base de datos. Se permite de manera explícita el almacenamiento en caché del contenido dinámico, dado que resulta muy útil para acelerar los sitios web. La prueba modela una librería electrónica y, como otras pruebas TPC, proporciona diferentes factores de escala. La métrica de rendimiento principal son las **interacciones web por segundo** (*web interactions per second*: WIPS) y el precio por WIPS. Sin embargo, la prueba TPC-W ya no se utiliza.

24.3. Otros temas sobre el desarrollo de aplicaciones

En esta sección se van a tratar dos temas sobre el desarrollo de aplicaciones: la prueba de las aplicaciones y su migración.

24.3.1. Prueba de aplicaciones

La prueba de programas implica el diseño de un **banco de pruebas,** es decir, una colección de casos de prueba. Las pruebas no son un proceso de una sola vez, ya que los programas evolucionan continuamente y los errores pueden aparecer como consecuencia imprevista de un cambio en el programa; este tipo de error se denomina **regresión** de programa. Por tanto, tras cada cambio en un programa, hay que volver a probarlo de nuevo. Normalmente no es factible que una persona se dedique a realizar las pruebas tras cada cambio; en vez de ello, se guardan los resultados esperados de las pruebas para cada caso de prueba en un banco de pruebas. Las **pruebas de regresión** implican volver a ejecutar el programa con cada uno de los casos de prueba del banco de pruebas, y comprobar si la salida que genera el programa es la esperada.

En el contexto de las aplicaciones de bases de datos, un caso de prueba consta de dos partes: un estado de la base de datos y una entrada a una determinada interfaz de la aplicación.

Las consultas de SQL pueden tener errores sutiles difíciles de encontrar. Por ejemplo, una consulta puede ejecutar $r \bowtie s$, cuando debería decir realmente $r ⟕ s$. La diferencia entre estas dos consultas solo se encontrará si la base de datos de prueba tiene una tupla en r que no casa con ninguna tupla de s. Por tanto, es importante crear bases de datos de prueba que permitan encontrar errores comunes. Estos errores se denominan **mutaciones,** ya que normalmente consisten en pequeños cambios en una consulta o programa. Un caso de prueba que genera una salida diferente de la esperada y una mutación de la consulta se conoce como **matar al mutante.** Un banco de prueba debería tener casos de prueba que maten a los mutantes (la mayoría) más comunes.

Si un caso de prueba realiza una actualización de la base de datos, para comprobar que se ha ejecutado correctamente hay que verificar que el contenido de la base de datos coincide con el esperado. Por tanto, el resultado esperado consta no solo de los datos que se muestren en la pantalla del usuario, sino también del estado de la base de datos (la actualización).

Como el estado de la base de datos puede ser muy grande, debería ser compartido por distintos casos de prueba. Las pruebas son complicadas dado que si un caso de prueba realiza una modificación en la base de datos, los resultados de otros casos de prueba que se ejecuten después en la misma base de datos puede que no coincidan con los resultados esperados. Los otros casos de prueba se podrían marcar, erróneamente, como fallidos. Para evitar este problema, siempre que un caso de prueba realice una actualización, se debe restaurar el estado de la base de datos a su estado original antes de ejecutar la prueba.

Las pruebas también se pueden utilizar para asegurar que la aplicación cumple con los requisitos de rendimiento. Para realizar estas **pruebas de rendimiento,** la base de datos de prueba debe tener el mismo tamaño que tendría la base de datos real. En algunos casos ya se dispone de datos sobre los que realizar la prueba. En otros casos se debe generar una base de datos del tamaño requerido; existen distintas herramientas para generar estas bases de datos de prueba. Estas herramientas aseguran que los datos generados satisfacen las restricciones de clave primaria y de clave externa. Adicionalmente pueden generar datos que parezcan tener cierto sentido, por ejemplo, rellenando un atributo nombre con nombres reales en lugar de con cadenas aleatorias. Algunas de estas herramientas también permiten especificar las distribuciones de los datos; por ejemplo, en una base de datos de una universidad se puede requerir que la mayoría de los estudiantes tengan edades entre los 18 y los 25 años, y la mayoría de los profesores entre los 25 y los 65 años.

Incluso aunque exista una base de datos, las organizaciones no suelen querer revelar datos sensibles a otras organizaciones externas para realizar las pruebas de rendimiento. Para estas situaciones, se puede realizar una copia de la base de datos real en la que se modifican los valores de la copia de forma que cualquier dato sensible, como los números de tarjeta de crédito, los números de la Seguridad Social o las fechas de nacimiento, se **ofusquen.** La ofuscación se realiza en la mayoría de los casos sustituyendo

el valor real por uno que se genera aleatoriamente, teniendo cuidado también de actualizar todas las referencias a dicho valor en el caso de que se trate de una clave primaria. Por otra parte, si la ejecución de la aplicación depende de dicho valor, como la fecha de nacimiento en una aplicación que realiza distintas acciones dependiendo de la fecha, la ofuscación puede realizar pequeños cambios aleatorios en los valores en lugar de sustituirlos completamente.

24.3.2. Migración de aplicaciones

Los **sistemas heredados** son sistemas de aplicaciones de generaciones anteriores en uso, pero que la organización desea sustituir por otros. Por ejemplo, muchas organizaciones desarrollaron aplicaciones, pero pueden decidir sustituirlas por productos comerciales. En algunos casos, un sistema heredado puede usar tecnologías antiguas incompatibles con las normas y sistemas de la generación actual. Algunos sistemas heredados operativos tienen varias décadas y se basan en tecnologías tales como bases de datos de red o jerárquicas, o usan Cobol y sistemas de ficheros sin bases de datos. Estos sistemas pueden contener datos valiosos y pueden dar soporte a aplicaciones esenciales.

La sustitución de las aplicaciones heredadas por aplicaciones más modernas suele resultar costosa tanto en términos de tiempo como de dinero, dado que suelen ser de tamaño muy grande, con millones de líneas de código desarrolladas por equipos de programadores a lo largo de varias décadas. Contienen grandes cantidades de datos que se deban trasladar a la nueva aplicación, que puede usar un esquema totalmente diferente. El cambio de la aplicación vieja a la nueva implica volver a formar a una gran cantidad de personal. El cambio se debe hacer generalmente sin ninguna interrupción, con los datos incorporados en el viejo sistema, también disponibles en el nuevo.

Muchas organizaciones intentan evitar reemplazar los sistemas heredados, intentando interoperar con los nuevos sistemas. Un enfoque usado para interoperar entre las bases de datos relacionales y las bases de datos heredadas es crear una capa, denominada **envoltura** (*wrapper*) por encima de los sistemas heredados que pueda hacer que el sistema heredado parezca una base de datos relacional. Puede que la envoltura ofrezca soporte para ODBC u otras normas de interconexión como OLE-DB, que puede usarse para consultar y actualizar el sistema heredado. La envoltura es responsable de la conversión de las consultas y actualizaciones relacionales en consultas y actualizaciones del sistema heredado.

Cuando una organización decide sustituir un sistema heredado por otro nuevo puede seguir un proceso denominado **ingeniería inversa,** que consiste en repasar el código del sistema heredado para obtener el diseño de los esquemas del modelo de datos requerido (como puede ser el modelo E-R o un modelo de datos orientado a los objetos). La ingeniería inversa también examina el código para averiguar los procedimientos y los procesos que se implementan con objeto de obtener un modelo de alto nivel del sistema. La ingeniería inversa es necesaria porque los sistemas heredados no suelen tener documentación de alto nivel de sus esquemas ni del diseño global de sus sistemas. Al iniciar el diseño de un nuevo sistema los desarrolladores repasan el diseño de modo que pueda mejorarse en lugar de volver a implementarlo tal como estaba. Se necesita una amplia labor de codificación para dar soporte a toda la funcionalidad (como pueden ser las interfaces de usuario y los sistemas de información) que proporcionaba el sistema heredado. El proceso completo se denomina **reingeniería.**

Cuando se ha creado y probado un sistema nuevo, hay que llenarlo con los datos del sistema heredado y todas las actividades posteriores se deben realizar en el sistema nuevo. No obstante, la transición abrupta al sistema nuevo, que se denomina **enfoque big-bang,** conlleva varios riesgos. En primer lugar, puede que los usuarios no estén familiarizados con las interfaces del nuevo sistema. En segundo lugar, puede haber fallos o problemas de rendimiento en el nuevo sistema que no se hayan descubierto al probarlo. Estos problemas pueden provocar grandes pérdidas a las empresas, dado que es posible que su capacidad para realizar transacciones críticas como las compras y las ventas resulte gravemente afectada. En algunos casos extremos se ha llegado a abandonar el sistema nuevo y se ha vuelto a usar el sistema heredado después de que fallara el intento de cambio.

Un enfoque alternativo, denominado **enfoque incremental,** sustituye la funcionalidad del sistema heredado de manera incremental. Por ejemplo, puede que las nuevas interfaces de usuario se utilicen con el sistema antiguo en el *back-end*, o viceversa. Otra opción es usar el sistema nuevo solo para algunas funcionalidades que puedan desgajarse del sistema heredado. En cualquier caso, los sistemas heredados y los nuevos coexisten durante algún tiempo. Por tanto, existe una necesidad de desarrollo y uso de envolturas del sistema heredado para proporcionar la funcionalidad requerida para interoperar con el sistema nuevo. Este enfoque, por tanto, tiene asociado un coste de desarrollo superior.

24.4. Normalización

Las **normas** definen las interfaces de los sistemas de software; por ejemplo, las normas definen la sintaxis y la semántica de los lenguajes de programación o las funciones en la interfaz de los programas de aplicaciones o, incluso, los modelos de datos (como las normas de las bases de datos orientadas a objetos). Actualmente los sistemas de bases de datos son complejos y suelen estar constituidos por varias partes creadas de manera independiente que deben interactuar entre sí. Por ejemplo, puede que los programas clientes se creen de manera independiente de los sistemas de *back-end*, pero todos ellos deben poder interactuar entre sí. Es posible que una empresa que tenga varios sistemas de bases de datos heterogéneos necesite intercambiar datos entre las bases de datos. En una situación de este tipo, las normas desempeñan un papel importante.

Las **normas formales** son normas desarrolladas por una organización de normalización o por grupos de empresas mediante un procedimiento público. Los productos dominantes se convierten a veces en una **norma de facto,** en el sentido de que resultan aceptados de manera general como normas sin necesidad de ningún procedimiento formal de reconocimiento. Algunas normas formales, como muchos aspectos de las normas SQL-92 y SQL:1999, son **normas anticipatorias** que lideran el mercado; definen las características que los fabricantes implementan posteriormente en los productos. En otros casos, las normas, o partes de las mismas, son **normas reaccionarias,** en el sentido de que intentan normalizar las características que ya han implementado algunos fabricantes, y que pueden haberse convertido, incluso, en normas de facto. SQL-89 era, en muchos sentidos, reaccionario, dado que estandarizaba características, como la comprobación de la integridad, que ya estaban presentes en la norma SAA SQL de IBM y en otras bases de datos.

Los comités de dictamen de normas formales suelen estar compuestos por representantes de los fabricantes y por miembros de grupos de usuarios y de organizaciones de normalización, como la Organización Internacional de Normalización (*International Organization for Standardization*: ISO) o el Instituto Nacional Estadounidense de Normalización (*American National Standards Institute*: ANSI), o de organismos profesionales como el Instituto de Ingenieros Eléctricos y Electrónicos (*Institute of Electrical and Electronics Engineers*: IEEE). Los comités para las normas formales se reúnen de manera periódica y sus componentes presentan

propuestas de características de la norma que hay que añadir o modificar. Tras un periodo de discusión (generalmente amplio), de modificaciones de la propuesta y de examen público, los integrantes de los comités votan la aceptación o el rechazo de las características. Tras cierto tiempo de haber definido e implementado una norma, quedan claras sus carencias y aparecen nuevas necesidades. El proceso de actualización de la norma comienza entonces, y tras unos cuantos años suele publicarse una nueva versión de la misma. Este ciclo suele repetirse cada pocos años hasta que, finalmente (quizás muchos años más tarde), la norma se vuelve tecnológicamente irrelevante o pierde su base de usuarios.

La norma CODASYL de DBTG para bases de datos en red, formulada por el Grupo de Trabajo para Bases de Datos (*Database Task Group*: DBTG) fue una de las primeras normas formales en este campo. Los productos de bases de datos de IBM solían establecer las normas de facto, dado que IBM ocupaba gran parte de este mercado. Con el crecimiento de las bases de datos relacionales aparecieron nuevos competidores en el negocio de las bases de datos; por tanto, surgió la necesidad de normas formales. En los últimos años, Microsoft ha creado varias especificaciones que también se han convertido en normas de facto. Un ejemplo destacable es ODBC, que se usa actualmente en entornos que no son de Microsoft. JDBC, cuya especificación fue creada por Sun Microsystems, es otra norma de facto muy usada.

Esta sección ofrece una introducción de muy alto nivel a las diferentes normas, concentrándose en los objetivos de cada norma. Las notas bibliográficas al final del capítulo ofrecen referencias de las descripciones detalladas de las normas mencionadas en esta sección.

24.4.1. Normas de SQL

Como SQL es el lenguaje de consultas más usado, se ha trabajado mucho en su normalización. ANSI e ISO, con los diferentes fabricantes de bases de datos, han desempeñado un papel protagonista en esta labor. La norma SQL-86 fue la versión inicial. La norma Arquitectura de Aplicaciones de Sistemas (*Systems Application Architecture*: SAA) de IBM para SQL se publicó en 1987. A medida que la gente identificaba la necesidad de más características se desarrollaron versiones actualizadas de la norma formal de SQL, denominadas SQL-89 y SQL-92.

La versión SQL:1999 de la norma SQL añadió varias características al lenguaje. Ya se han visto muchas de estas características en los capítulos anteriores. La versión SQL:2003 de la norma SQL es una extensión menor de la norma SQL:1999. Algunas de las características OLAP (Sección 5.6.3) de SQL:1999 se especificaron como una enmienda en lugar de esperar a la versión de SQL:2003.

La norma SQL:2003 se ha dividido en varias partes:

- Parte 1: SQL/Framework proporciona una introducción a la norma.
- Parte 2: SQL/Foundation define los fundamentos de la norma: tipos, esquemas, tablas, vistas, consultas y sentencias de actualización, expresiones, modelo de seguridad, predicados, reglas de asignación, gestión de transacciones, etc.
- Parte 3: SQL/CLI (*Call Level Interface*) define las interfaces de los programas de aplicaciones con SQL.
- Parte 4: SQL/PSM (*Persistent Stored Modules*) define las extensiones de SQL para hacerlo procedimental.
- Parte 9: SQL/MED (*Management of External Data*) define las normas para la realización de interfaces en los sistemas SQL con orígenes externos. Al escribir envolturas, los diseñadores de los sistemas pueden tratar los orígenes externos de datos, como pueden ser los archivos o los datos de bases de datos no relacionales, como si fueran tablas «externas».
- Parte 10: SQL/OLB (*Object Language Bindings*) define las normas para la incrustación de SQL en Java.
- Parte 11: SQL/Schemata (*Information and Definition Schema*) define una interfaz estándar para el catálogo.
- Parte 13: SQL/JRT (*Java Routines and Types*) define una norma para el acceso a rutinas y tipos de Java.
- Parte 14: SQL/XML define especificaciones relacionadas con XML.

Los números que faltan tratan características como los datos temporales, el procesamiento de transacciones distribuidas y los datos multimedia, para las que todavía no existe ningún acuerdo normativo.

Las versiones más recientes de SQL son SQL:2006, que añade varias características relacionadas con XML, y SQL:2008, que incluye varias extensiones al lenguaje.

24.4.2. Normas de conectividad de las bases de datos

La norma **ODBC** es una norma muy usada para la comunicación entre aplicaciones de clientes y los sistemas de bases de datos. ODBC se basa en las normas SQL de **interfaz de nivel de llamada** (*call level interface*: CLI) desarrolladas por el consorcio industrial **X/Open** y el Grupo SQL Access, pero tienen varias extensiones. La API ODBC define una CLI, una definición de sintaxis SQL y reglas sobre las secuencias admisibles de llamadas CLI. La norma también define los niveles de conformidad para la CLI y la sintaxis SQL. Por ejemplo, el nivel del núcleo de la CLI tiene comandos para conectarse con bases de datos, para preparar y ejecutar sentencias SQL, para devolver resultados o valores de estado y para administrar transacciones. El siguiente nivel de conformidad (nivel uno) exige el soporte de la recuperación de información de los catálogos y otras características que superan la CLI del nivel del núcleo; el nivel dos exige más características, como la capacidad de enviar y recuperar arrays de valores de parámetros e información de catálogo más detallada.

ODBC permite que un cliente se conecte de manera simultánea con varios orígenes de datos y que conmute entre ellos, pero las transacciones en cada uno de ellos son independientes entre sí; ODBC no soporta el compromiso de dos fases.

Los sistemas distribuidos ofrecen un entorno más general que los sistemas cliente–servidor. El consorcio X/Open también ha desarrollado las **normas X/Open XA** para la interoperabilidad de las bases de datos. Estas normas definen las primitivas de gestión de las transacciones (como pueden ser el comienzo de las transacciones, su compromiso, su anulación y su preparación para el compromiso) que deben proporcionar las bases de datos que cumplan con la norma; los administradores de las transacciones pueden invocar estas primitivas para implementar las transacciones distribuidas mediante compromiso de dos fases. Las normas XA son independientes de los modelos de datos y de las interfaces concretas entre los clientes y las bases de datos para el intercambio de datos. Por tanto, se pueden usar los protocolos XA para implementar sistemas de transacciones distribuidas en los que una sola transacción pueda tener acceso a bases de datos relacionales y orientadas a objetos, y que el administrador de transacciones asegure la consistencia global mediante el compromiso de dos fases.

Hay muchos orígenes de datos que no son bases de datos relacionales y, de hecho, puede que no sean ni siquiera bases de datos. Ejemplos de ello son los archivos planos y los correos electrónicos. **OLE-DB,** de Microsoft, es una API de C++ con objetivos parecidos a los de ODBC, pero para orígenes de datos que no son bases de datos y que puede que solo proporcionen servicios limitados de consulta y de actualización. Al igual que ODBC, OLE-DB proporcio-

na estructuras para la conexión con orígenes de datos, el inicio de una sesión, la ejecución de comandos y la devolución de resultados en forma de conjunto de filas, que es un conjunto de filas de resultados.

Sin embargo, OLE-DB se diferencia de ODBC en varios aspectos. Para dar soporte a los orígenes de datos con soporte de características limitadas, las características de OLE-DB se dividen entre varias interfaces y cada origen de datos solo puede implementar un subconjunto de esas interfaces. Los programas OLE-DB pueden negociar con los orígenes de datos para averiguar las interfaces que soportan. En ODBC los comandos siempre están en SQL. En OLE-DB los comandos pueden estar en cualquier lenguaje soportado por el origen de datos; aunque puede que algunos orígenes de datos soporten SQL, o un subconjunto limitado de SQL, puede que otros orígenes solo ofrezcan posibilidades sencillas, como el acceso a los datos de los archivos planos, sin ninguna capacidad de consulta. Otra diferencia importante de OLE-DB con respecto a ODBC es que los conjuntos de filas son objetos que pueden compartir varias aplicaciones mediante la memoria compartida. Cuando una aplicación actualiza objetos conjuntos de filas, a las otras aplicaciones que compartan ese objeto se les notificará la modificación.

La API **Objetos Activos de Datos** (*Active Data Objects*: ADO), también creada por Microsoft, ofrece una interfaz sencilla de usar con la funcionalidad OLE-DB que puede llamarse desde los lenguajes de guiones, como VBScript y JScript. La API más reciente, **ADO.NET,** se ha diseñado para las aplicaciones escritas en los lenguajes .NET; por ejemplo, C# y Visual Basic.NET. Además de proporcionar interfaces simplificadas, proporciona una abstracción llamada *DataSet* que permite el acceso desconectado a los datos.

24.4.3. Normas de las bases de datos de objetos

Hasta ahora las normas en el área de las bases de datos orientadas a objetos han sido impulsadas fundamentalmente por los fabricantes de BDOO. El **Grupo de Gestión de Bases de Datos de Objetos** (*Object Database Management Group*: ODMG) fue un grupo formado por fabricantes de BDOO para normalizar los modelos de datos y las interfaces de lenguaje con las BDOO. En el Capítulo 22 se describió brevemente la interfaz del lenguaje C++ especificada por ODMG. ODMG ya no está activo, JDO es una norma para añadir persistencia a Java.

El **Grupo de Administración de Objetos** (*Object Management Group*: OMG) es un consorcio de empresas formado con el objetivo de desarrollar una arquitectura estándar para las aplicaciones de software distribuido basadas en el modelo orientado a objetos. OMG creó el modelo de referencia **arquitectura de gestión de objetos** (*Object Management Architecture*: OMA). El **agente para solicitudes de objetos** (*Object Request Broker*: ORB) es un componente de la arquitectura OMA que proporciona de manera transparente la entrega de mensajes a los objetos distribuidos, de modo que la ubicación física del objeto no tiene importancia. La **arquitectura común de agente para solicitudes de objetos** (*Common Object Request Broker Architecture*: CORBA) proporciona una especificación detallada de ORB, e incluye un **lenguaje de descripción de interfaces** (*interface description language*: IDL), que se usa para definir los tipos de datos usados para el intercambio de datos. IDL ayuda a dar soporte a la conversión de datos cuando estos se intercambian entre sistemas con diferentes representaciones de los mismos.

Microsoft introdujo el **modelo datos-entidades,** que incorpora ideas de los modelos entidad-relación, de datos orientados a objetos y también un enfoque de integración de consultas con los lenguajes de programación, denominado **consultas integradas en el lenguaje** (*language integrates querying*: LINQ). Probablemente se coviertan en normas de facto.

24.4.4. Normas basadas en XML

Se ha definido una amplia variedad de normas basadas en XML (consulte el Capítulo 23) para gran número de aplicaciones. Muchas de estas normas están relacionadas con el comercio electrónico. Entre ellas hay normas promulgadas por consorcios sin ánimo de lucro y se han realizado grandes esfuerzos por parte de las empresas para crear normas de facto.

RosettaNet, que pertenece a la primera categoría, es un consorcio industrial que usa normas basadas en XML para facilitar la gestión de las cadenas de abastecimiento en las industrias informáticas y de tecnologías de la información. Las cadenas de abastecimiento se refieren a las compras del material y de los servicios que una organización necesita para funcionar. En cambio, la gestión de la relación de los clientes se refiere a la parte visible de la interacción con los clientes de una empresa. Las cadenas de abastecimiento requieren la normalización de varias cosas, tales como:

- **Identificador global de compañías.** Especifica un sistema para identificar unívocamente a las compañías, usando un identificador de nueve dígitos denominado *Sistema de Numeración Universal de Datos* (*Data Universal Numbering System*: DUNS).
- **Identificador global de productos.** Especifica el número de catorce dígitos *Número Global de Productos Comerciales* (*Global Trade Item Number*: GTIN) para identificar productos y servicios.
- **Identificador global de clases.** Se trata de un código jerárquico de diez dígitos para clasificar productos y servicios denominado *Código Estándar de Productos y Servicios de las Naciones Unidas* (United Nations/Standard Product and Services Code: UN/SPSC).
- **Interfaces entre los socios implicados.** Los *Procesos de Interfaz entre Socios* (*Partner Interface Processes*: PIP) de RosettaNet definen procesos de negocio entre socios. Estos procesos son diálogos entre sistemas basados en XML: definen los formatos y la semántica de los documentos de negocio implicados en el proceso, así como los pasos necesarios para completar una transacción. Algunos ejemplos de pasos serían conseguir la información de producto y de servicio; los pedidos de compras; la facturación de los pedidos; los pagos; las peticiones del estado de los pedidos; la gestión de inventarios y el soporte posventa, incluyendo la garantía del servicio, etc. El intercambio de la información sobre el diseño, la configuración, los procesos y la calidad también es posible para coordinar las actividades de fabricación entre diferentes organizaciones.

Los participantes en los mercados electrónicos pueden guardar los datos en gran variedad de sistemas de bases de datos. Estos sistemas pueden utilizar diferentes modelos, formatos y tipos de datos. Además, puede que haya diferencias semánticas (sistema métrico frente a sistema imperial británico, distintas divisas, etc.) en los datos. Las normas para los mercados electrónicos incluyen los métodos para *envolver* cada uno de estos sistemas heterogéneos con un esquema XML. Estas *envolturas* XML forman la base de una vista unificada de los datos para todos los participantes del mercado.

El **Protocolo Simple de Acceso a Objetos** (*Simple Object Access Protocol*: SOAP) es una norma para llamadas a procedimientos remotos que usa XML para codificar los datos (tanto los parámetros como los resultados) y usa HTTP como protocolo de transporte; es decir, las llamadas a procedimientos se transforman en solicitudes HTTP. Está apoyado por el Consorcio World Wide Web (*World Wide Web Consortium*: W3C) y ha logrado una amplia aceptación en la industria. Puede usarse en gran variedad de aplicaciones; por ejemplo, en el comercio electrónico entre empresas, las aplicaciones que se ejecutan en un sitio pueden tener acceso a los datos de otros sitios y ejecutar acciones mediante SOAP.

SOAP y los servicios web se tratan con más detalle en la Sección 23.7.3.

24.5. Resumen

- El ajuste de los parámetros del sistema de bases de datos, así como el diseño de nivel superior de la base de datos, como pueden ser el esquema, los índices y las transacciones, resultan importantes para lograr un buen rendimiento. Las consultas se pueden ajustar para mejorar la orientación de conjuntos, mientras que las utilidades de carga masiva permiten mejorar de forma considerable la importación de datos.

  La mejor forma de realizar el ajuste es identificar los cuellos de botella y eliminarlos. Los sistemas de bases de datos suelen tener distintos parámetros ajustables, como los tamaños de memoria intermedia, el tamaño de memoria y el número de discos. Se pueden elegir apropiadamente los conjuntos de índices y las vistas materializadas para minimizar los costes totales. Se pueden ajustar las transacciones para minimizar la contención de bloqueos; las funciones de aislamiento de instantáneas y numeración secuencial con liberación temprana de bloqueos son herramientas útiles para reducir la contención de lectura-escritura y de escritura-escritura.
- Las pruebas de rendimiento desempeñan un papel importante en las comparaciones entre sistemas de bases de datos, especialmente a medida que cumplen cada vez más las normas. El conjunto de pruebas TPC se usa mucho y las diferentes pruebas TPC resultan útiles para comparar el rendimiento de las bases de datos con diferentes cargas de trabajo.
- En necesario probar extensamente las aplicaciones según se desarrollan y antes de su despliegue. Las pruebas se utilizan para encontrar errores así como para asegurar que se cumplen los objetivos de rendimiento.
- Los sistemas heredados son sistemas basados en tecnologías de generaciones anteriores como las bases de datos no relacionales o, incluso, directamente en sistemas de archivos. Suele ser importante el establecimiento de las interfaces entre los sistemas heredados y los sistemas de última generación cuando ejecutan sistemas de misión crítica. La migración desde los sistemas heredados a los sistemas de última generación debe realizarse con cuidado para evitar interrupciones, que pueden resultar muy costosas.
- Las normas son importantes debido a la complejidad de los sistemas de bases de datos y a la necesidad de interoperatividad. Hay normas formales para SQL. Las normas de facto, como ODBC y JDBC, y las normas adoptadas por grupos de empresas, como CORBA, han desempeñado un papel importante en el crecimiento de los sistemas de bases de datos cliente-servidor.

Términos de repaso

- Ajuste del rendimiento.
- Orientación del conjunto.
- Actualización en lotes (JDBC).
- Carga masiva.
- Actualización masiva.
- Sentencia merge.
- Cuellos de botella.
- Sistemas de colas.
- Parámetros ajustables.
- Ajuste del hardware.
- Regla de los cinco minutos.
- Regla del minuto.
- Ajuste del esquema.
- Ajuste de los índices.
- Vistas materializadas.
- Mantenimiento inmediato de vistas.
- Mantenimiento diferido de vistas.
- Ajuste de las transacciones.
- Contención de bloqueos.
- Secuencias.
- Procesamiento de transacciones por minilotes.
- Simulación del rendimiento.
- Pruebas de rendimiento.
- Tiempo de servicio.
- Tiempo hasta finalización.
- Clases de aplicaciones de bases de datos.
- Pruebas TPC.
  - TPC-A.
  - TPC-B.
  - TPC-C.
  - TPC-D.
  - TPC-E.
  - TPC-H.
- Interacciones web por segundo.
- Pruebas de regresión.
- Matar mutantes.
- Sistemas heredados.
- Ingeniería inversa.
- Reingeniería.
- Normalización.
  - Normas formales.
  - Normas de facto.
  - Normas anticipatorias.
  - Normas reaccionarias.
- Normas de conectividad para bases de datos.
  - ODBC.
  - OLE-DB.
  - Normas X/Open XA.
- Normas de bases de datos de objetos.
  - ODMG.
  - CORBA.
- Normas basadas en XML.

Ejercicios prácticos

24.1. Muchas aplicaciones necesitan generar los números de secuencia para cada transacción.

a. Si un contador de secuencia se bloquea en dos fases, puede producirse un cuello de botella en la concurrencia. Explique por qué puede ocurrir esto.

b. Muchos sistemas de bases de datos soportan contadores de secuencia predefinidos, que no se bloquean en dos fases; cuando una transacción solicita un número de secuencia, se bloquea el contador, se incrementa y se desbloquea.

i Explique cómo pueden mejorar la concurrencia estos contadores.

ii Explique por qué puede haber saltos en los números de secuencia del conjunto final de transacciones comprometidas.

24.2. Suponga una relación $r(a, b, c)$.

a. Indique un ejemplo de una situación bajo la cual el rendimiento de las consultas de selección con igualdad sobre el atributo a pueda verse seriamente afectada dependiendo de cómo se agrupe r.

b. Suponga que también se tengan consultas de selección de rangos sobre el atributo b. ¿Se puede agrupar r de forma que las consultas de selección con igualdad sobre $r.a$ y las de rango sobre $r.b$ puedan resolverse eficientemente?

Justifique la respuesta.

c. Si no es posible el agrupamiento anterior, sugiera cómo se pueden ejecutar eficientemente ambos tipos de preguntas eligiendo índices apropiados, suponiendo que la base de datos solo soporta planes con índices (es decir, si toda la información requerida para una consulta está disponible en un índice, la base de datos puede generar un plan que utilice el índice pero sin acceder a la relación).

24.3. Suponga que una aplicación de la base de datos parece no tener cuellos de botella; es decir, la CPU y el uso de disco son altos y prácticamente todas las consultas de la base de datos están equilibradas. ¿Significa esto que la aplicación no se puede ajustar mejor? Justifique la respuesta.

24.4. Suponga que un sistema ejecuta tres tipos de transacciones. Las transacciones de tipo A se ejecutan a razón de 50 por segundo, las transacciones de tipo B se ejecutan a 100 por segundo y las transacciones de tipo C se ejecutan a 200 por segundo. Suponga que la mezcla de transacciones tiene un 25 por ciento del tipo A, otro 25 por ciento del tipo B y un 50 por ciento del tipo C.

a. ¿Cuál es el rendimiento promedio de transacciones del sistema, suponiendo que no hay interferencia entre las transacciones?

b. ¿Qué factores pueden generar interferencias entre las transacciones de los diferentes tipos, haciendo que la productividad calculada sea incorrecta?

24.5. Indique algunas ventajas e inconvenientes de las normas anticipatorias frente a las normas reaccionarias.

Ejercicios

24.6. Determine toda la información de rendimiento que proporciona su sistema preferido de bases de datos. Busque por lo menos lo siguiente: las consultas que se están ejecutando actualmente o se han ejecutado recientemente, los recursos que consumió (CPU y E/S) cada una de ellas, la fracción de peticiones que generaron fallos de página en la memoria intermedia (para cada consulta, si está disponible) y los bloqueos que tienen un alto grado de contención. También se puede conseguir la información sobre el uso de la CPU y de E/S del sistema operativo.

24.7. a. ¿Cuáles son los tres niveles principales en los que se puede ajustar un sistema de bases de datos para mejorar su rendimiento?

b. Proponga dos ejemplos del modo en que se puede realizar el ajuste para cada uno de los niveles.

24.8. Al realizar un ajuste de rendimiento, ¿se ajustaría primero el hardware (añadiendo discos o memoria), o las transacciones (añadiendo índices o vistas materializadas)?

Justifique la respuesta.

24.9. Suponga que su aplicación tiene transacciones cada una de las cuales con acceso y actualización a una única tupla en una relación muy grande almacenada en una organización de archivos de árbol B^+. Suponga que todos los nodos internos del árbol B^+ están en la memoria, pero solamente una fracción muy pequeña de las páginas hoja puede caber en la memoria. Explique cómo calcular el número mínimo de discos necesarios para permitir una carga de trabajo de 1.000 transacciones por segundo. Calcule también el número necesario de discos usando los valores de los parámetros de disco de la Sección 10.2.

24.10. ¿Cuál es el motivo para dividir una transacción de larga duración en una serie de transacciones más cortas? ¿Qué problemas pueden surgir como consecuencia y cómo pueden evitarse?

24.11. Suponga que el precio de la memoria cae a la mitad y la velocidad de acceso al disco (número de accesos por segundo) se dobla mientras el resto de los factores permanecen iguales. ¿Cuál sería el efecto de este cambio en las reglas de los cinco minutos y del minuto?

24.12. Indique al menos cuatro de las características de las pruebas TPC que ayudan a hacer las medidas realistas y dignas de confianza.

24.13. ¿Por qué se sustituyó TPC-D por las pruebas de rendimiento TPC-H y TPC-R?

24.14. Explique las características de la aplicación que ayudarían a decidir entre TPC-C, TPC-H o TPC-R para modelar mejor la aplicación.

Notas bibliográficas

Un texto clásico sobre teoría de colas es el de Kleinrock [1975].

Una de las primeras propuestas de pruebas de sistemas de bases de datos (la prueba Wisconsin) la realizaron Bitton et ál. [1983]. Las pruebas TPC-A, -B y -C se describen en Gray [1991]. Una versión en línea de todas las descripciones de las pruebas TPC, así como de los resultados de estas pruebas, está disponible en World Wide Web en la URL www.tpc.org; el sitio también contiene información actualizada sobre nuevas propuestas de pruebas. La prueba OO1 para OODBs se describe en Cattell y Skeen [1992]; la prueba OO7 se describe en Carey et ál. [1993].

Shasha y Bonnet [2002] ofrecen un tratamiento detallado del ajuste de bases de datos. O'Neil y O'Neil [2000] ofrecen un tratamiento de libro de texto de muy buena calidad sobre la medida del rendimiento y su ajuste. Las reglas de los cinco minutos y del minuto se describen en Gray y Putzolu [1987], y más recientemente se ha extendido para considerar combinaciones de memoria principal, memorias flash y discos en Graefe [2008].

La selección de índices y de vistas materializadas se abordan en Ross et ál. [1996], Chaudhuri y Narasayya [1997], Agrawal et ál. [2000] y en Mistry et ál. [2001]. En Zilio et ál. [2004], Dageville et ál. [2004] y Agrawal et ál. [2004] se describe el ajuste en DB2 de IBM, Oracle y SQL Server de Microsoft.

Se puede hallar información sobre ODBC, OLE-DB, ADO y ADO.NET en el sitio web www.microsoft.com/data y en varios libros sobre el tema que pueden encontrarse en www.amazon.com. La revista *Sigmod Record*, de *ACM*, que se publica trimestralmente, tiene una sección fija sobre las normas de bases de datos.

Se halla disponible en línea gran cantidad de información sobre las normas basadas en XML en el sitio web www.w3c.org. La información sobre RosettaNet se puede encontrar en el sitio web www.rosettanet.org.

La reingeniería de procesos de negocio se trata en Cook [1996]. Umar [1997] trata la reingeniería y aspectos del trabajo con sistemas heredados.

Datos espaciales, temporales y movilidad

25

Durante la mayor parte de la historia de las bases de datos, sus tipos de datos eran relativamente sencillos, y esto se reflejó en las primeras versiones de SQL. En los últimos años, sin embargo, ha habido una necesidad creciente de manejar tipos de datos nuevos en las bases de datos, como los datos temporales, los datos espaciales y los datos multimedia.

Otra tendencia importante de la última década ha creado sus propios problemas: el auge de la informática móvil, que comenzó con las computadoras portátiles y las agendas de bolsillo, se ha extendido en tiempos más recientes a los teléfonos móviles con capacidad de procesamiento así como a una gran variedad de computadoras portátiles, que se usan cada vez más en aplicaciones comerciales.

En este capítulo se estudian varios tipos nuevos de datos y también los problemas de las bases de datos con estas aplicaciones.

25.1. Motivación

Antes de abordar a fondo cada uno de los temas se resumirá la motivación de estos tipos de datos y algunos problemas importantes del trabajo con ellos.

- **Datos temporales.** La mayor parte de los sistemas de bases de datos modelan un estado actual del mundo; por ejemplo, los clientes actuales, los estudiantes actuales y los cursos que se están ofertando. En muchas aplicaciones es muy importante almacenar y recuperar la información sobre estados anteriores. La información histórica puede incorporarse de manera manual en el diseño del esquema. No obstante, la tarea se simplifica mucho con el soporte de los datos temporales en las bases de datos, que se estudia en la Sección 25.2.
- **Datos espaciales.** Entre los datos espaciales están los **datos geográficos,** como los mapas y la información asociada a ellos, y los **datos de diseño asistido por computadora,** como los diseños de circuitos integrados o los diseños de edificios. Las aplicaciones de datos espaciales en un principio almacenaban los datos como archivos de un sistema de archivos, igual que las aplicaciones de negocios de las primeras generaciones. Pero, a medida que la complejidad y el volumen de los datos, y el número de usuarios, han aumentado, los enfoques ad hoc para almacenar y recuperar los datos en un sistema de archivos se han mostrado insuficientes para cubrir las necesidades de muchas aplicaciones que usan datos espaciales.

  Las aplicaciones de datos espaciales necesitan los servicios ofrecidos por los sistemas de bases de datos, en especial la posibilidad de almacenar y consultar de manera eficiente grandes cantidades de datos. Algunas aplicaciones también pueden necesitar otras características de las bases de datos, como las actualizaciones atómicas de parte de los datos almacenados, la durabilidad y el control de concurrencia. En la Sección 25.3 se estudian las ampliaciones de los sistemas de bases de datos tradicionales necesarias para que soporten los datos espaciales.
- **Datos multimedia.** En la Sección 25.4 se estudian las características necesarias en los sistemas de bases de datos que almacenan datos multimedia como los datos de imágenes, de vídeo o de audio. La principal característica que diferencia a los datos de vídeo y de audio es que la visualización de estos exige la recuperación a una velocidad predeterminada constante; por tanto, esos datos se denominan **datos de medios continuos.**
- **Bases de datos móviles.** En la Sección 25.5 se estudian los requisitos para las bases de datos de la nueva generación de sistemas computacionales móviles, como las computadoras portátiles y los teléfonos inteligentes, que se conectan con las estaciones base mediante redes de comunicación digitales inalámbricas. Estas computadoras necesitan poder operar mientras se hallan desconectadas de la red, a diferencia de los sistemas distribuidos de bases de datos estudiados en el Capítulo 19. También tienen una capacidad de almacenamiento limitada y, por tanto, necesitan técnicas especiales para la administración de la memoria.

25.2. El tiempo en las bases de datos

Las bases de datos modelan el estado de algunos aspectos del mundo real. Generalmente, las bases de datos solo modelan un estado, el estado actual del mundo real, y no almacenan información sobre estados anteriores, salvo posiblemente como registro de auditoría. Cuando se modifica el estado del mundo real, se actualiza la base de datos y se pierde la información sobre el estado anterior. Sin embargo, en muchas aplicaciones es importante almacenar y recuperar información sobre estados anteriores. Por ejemplo, una base de datos sobre pacientes debe almacenar información sobre el historial médico de cada paciente. El sistema de control de una fábrica puede almacenar información sobre las lecturas actuales y anteriores de los sensores de la fábrica para su análisis. Las bases de datos que almacenan información sobre los estados del mundo real a lo largo del tiempo se denominan **bases de datos temporales.**

Al considerar el problema del tiempo en los sistemas de bases de datos se distingue entre el tiempo medido por el sistema y el tiempo observado en el mundo real. El **tiempo válido** para un hecho es el conjunto de intervalos de tiempo en el que el hecho es cierto en el mundo real. El **tiempo de transacción** de un hecho es el intervalo de tiempo en el que el hecho es actual en el sistema de bases de datos. Este segundo tiempo se basa en el orden de secuenciación de las transacciones y el sistema lo genera de manera automática. Hay que tener en cuenta que los intervalos de tiempo válidos, que son un concepto del tiempo real, no pueden generarse de manera automática y deben proporcionarse al sistema.

Una **relación temporal** es una relación en la que cada tupla tiene un tiempo asociado en que es verdadera; el tiempo puede ser la hora real o el tiempo asociado a la transacción. Por supuesto, ambos pueden almacenarse, en cuyo caso la relación se denomina **relación bitemporal.** La Figura 25.1 muestra un ejemplo de relación temporal. Para simplificar la representación cada tupla tiene asociado un único intervalo temporal; así, cada tupla se representa una vez para cada intervalo de tiempo disjunto en que es cierta. Los intervalos se muestran aquí como pares de atributos *de* y *a*; una implementación real tendría un tipo estructurado, acaso denominado *Intervalo*, que contuviera los dos campos. Observe que algunas de la tuplas tienen un asterisco («*») en la columna temporal *a*; esos asteriscos indican que la tupla es verdadera hasta que se modifique el valor de la columna temporal *a*; así, la tupla es cierta en el momento actual. Aunque los tiempos se muestran en forma textual, se almacenan internamente de una manera más compacta, como el número de segundos desde algún momento de una fecha fija (como las 12:00 a. m., 1 de enero, 1900) que puede volver a traducirse a la forma textual normal.

ID	*nombre*	*nombre_dept*	*sueldo*	*de*	*a*
10101	Srinivasan	Informática	61000	2007/1/1	2007/12/31
10101	Srinivasan	Informática	65000	2008/1/1	2008/12/31
12121	Wu	Finanzas	82000	2005/1/1	2006/12/31
12121	Wu	Finanzas	87000	2007/1/1	2007/12/31
12121	Wu	Finanzas	90000	2008/1/1	2008/12/31
98345	Kim	Electrónica	80000	2005/1/1	2008/12/31

Figura 25.1. La relación temporal *profesor*.

25.2.1. Especificación del tiempo en SQL

El estándar SQL define los tipos **date, time** y **timestamp.** El tipo **date** contiene cuatro cifras para los años (1–9999), dos para los meses (1–12) y dos más para los días (1–31). El tipo **time** contiene dos cifras para la hora, dos para los minutos y otras dos para los segundos, además de cifras decimales opcionales. El campo de los segundos puede superar el valor de sesenta para tener en cuenta los segundos extra que se añaden algunos años para corregir las pequeñas variaciones de velocidad en la rotación de la Tierra. El tipo **timestamp** contiene los campos de **date** y de **time** con seis cifras decimales para el campo de los segundos.

Como las diferentes partes del mundo tienen horas locales diversas, suele necesitarse especificar la zona horaria junto con la hora. La **hora universal coordinada** (*universal coordinated time*: UTC) es un punto estándar de referencia para la especificación de la hora, y las horas locales se definen como diferencias respecto a la UTC. (La abreviatura estándar es UTC, en lugar de UCT —*Universal Coordinated Time*, en inglés— ya que es una abreviatura de *universal coordinated time*», escrito en francés como *universel temps coordonné*). SQL también soporta dos tipos: **time with time zone** y **timestamp with time zone,** que especifican la hora como la hora local más la diferencia de la hora local respecto a la UTC. Por ejemplo, la hora podría expresarse en términos de la hora estándar del este de EE. UU. (*U. S. Eastern Standard Time*), con una diferencia de −6:00, ya que la hora estándar del este de Estados Unidos va retrasada seis horas respecto de la UTC.

SQL soporta un tipo denominado **interval,** que permite hacer referencia a un periodo de tiempo como puede ser «1 día» o «2 días y 5 horas», sin especificar la hora concreta en que comienza el periodo. Este concepto se diferencia del concepto de intervalo usado anteriormente, que hace referencia a un intervalo de tiempo con horas de comienzo y de final específicas.[1]

1 Muchos investigadores de bases de datos temporales consideran que este tipo de datos debería haberse denominado **span,** ya que no especifica un momento inicial ni un momento final exactos, sino tan solo el tiempo transcurrido entre los dos.

25.2.2. Lenguajes de consultas temporales

Las relaciones de bases de datos sin información temporal a veces se denominan **relaciones instantáneas,** ya que reflejan el estado del mundo real en una instantánea. Así, una instantánea de una relación temporal en un momento del tiempo t es el conjunto de tuplas de la relación que son ciertas en el momento t, con los atributos de intervalos de tiempo eliminados. La operación instantánea para una relación temporal da la instantánea de la relación en un momento especificado, o en el momento actual si no se especifica el momento.

Una **selección temporal** es una selección que implica a los atributos de tiempo; una **proyección temporal** es una proyección en la que las tuplas de la proyección heredan los valores de tiempo de las tuplas de la relación original. Una **reunión temporal** es una reunión en la que los valores temporales de las tuplas del resultado son la intersección de los valores temporales de las tuplas de las que proceden. Si los valores temporales no se intersecan, esas tuplas se eliminan del resultado.

Los predicados *precede*, *solapa* y *contiene* pueden aplicarse a los intervalos; sus significados deben quedar claros. La operación *intersección* puede aplicarse a dos intervalos, para dar un único intervalo (posiblemente vacío). Sin embargo, la unión de dos intervalos puede que no sea un solo intervalo.

En las relaciones temporales las dependencias funcionales deben usarse con cuidado, como se vio en la Sección 8.9. Aunque el *ID* de profesor determine funcionalmente el sueldo en cierto momento, evidentemente este puede cambiar con el tiempo. Una **dependencia funcional temporal** $X \xrightarrow{T} Y$ se cumple para el esquema de relación R si, para todos los ejemplares legales r de R, todas las instantáneas de r satisfacen la dependencia funcional $X \rightarrow Y$.

Se han realizado varias propuestas de ampliación de SQL para mejorar su soporte de datos temporales, pero al menos hasta SQL:2008 no se ha proporcionado ningún soporte especial para los datos temporales aparte de los tipos de datos relativos a la fecha y hora y sus operaciones.

25.3. Datos espaciales y geográficos

El soporte de los datos espaciales en las bases de datos es importante para el almacenaje, indexado y consulta eficientes de los datos basados en las posiciones espaciales. Por ejemplo, suponga que se desea almacenar un conjunto de polígonos en una base de datos y consultar la base de datos para hallar todos los polígonos que intersecan un polígono dado. No se pueden usar las estructuras estándares de índices como los árboles B o los índices asociativos para responder de manera eficiente esas consultas. El procesamiento eficiente de esas consultas necesita estructuras de índices de finalidades especiales, como los árboles R (que se estudiarán posteriormente).

Dos tipos de datos espaciales son especialmente importantes:

- Los **datos de diseño asistido por computadora** (*computer aided design*: CAD), que incluyen información espacial sobre el modo en que los objetos, como edificios, coches o aviones, están construidos. Otros ejemplos importantes de bases de datos de diseño asistido por computadora son los diseños de circuitos integrados y de dispositivos electrónicos.
- Los **datos geográficos,** como los mapas de carreteras, los mapas de uso de la tierra, los mapas topográficos, los mapas políticos que muestran fronteras, los mapas catastrales, etc. Los **sistemas de información geográfica** son bases de datos de propósito especial adaptadas para el almacenamiento de datos geográficos.

El soporte para los datos geográficos se ha añadido a muchos sistemas de bases de datos, como IBM DB2 Spatial Extender, Informix Spatial Datablade u Oracle Spatial.

25.3.1. Representación de la información geométrica

La Figura 25.2 muestra el modo en que se pueden representar de manera normalizada en las bases de datos varias estructuras geométricas.

Hay que destacar aquí que la información geométrica puede representarse de varias maneras diferentes, de las que solo se describen algunas.

Un *segmento rectilíneo* puede representarse mediante las coordenadas de sus extremos. Por ejemplo, en una base de datos de mapas, las dos coordenadas de un punto serían su latitud y su longitud. Una **línea poligonal** (también denominada **línea quebrada**) consiste en una secuencia conectada de segmentos rectilíneos, y puede representarse mediante una lista que contenga las coordenadas de los extremos de los segmentos, en secuencia.

Se puede representar aproximadamente una curva arbitraria mediante líneas poligonales, dividiendo la curva en una serie de segmentos. Esta representación resulta útil para elementos bidimensionales como las carreteras; en este caso, el ancho de la carretera es lo bastante pequeño en relación con el tamaño de todo el mapa que puede considerarse bidimensional.

Algunos sistemas también soportan como primitivas los *arcos de circunferencia*, lo que permite que las curvas se representen como secuencias de arcos.

Los *polígonos* pueden representarse indicando sus vértices en orden, como en la Figura 25.2.[2] La lista de los vértices especifica la frontera de cada región poligonal. En una representación alternativa cada polígono puede dividirse en un conjunto de triángulos, como se muestra en la Figura 25.2. Este proceso se denomina **triangulación,** y se puede triangular cualquier polígono. Se concede un identificador a cada polígono complejo, y cada uno de los triángulos en los que se divide lleva el identificador del polígono. Los círculos y las elipses pueden representarse por los tipos correspondientes, o aproximarse mediante polígonos.

Las representaciones de las líneas poligonales o de los polígonos basadas en listas suelen resultar convenientes para el procesamiento de consultas. Esas representaciones que no están en la primera forma normal se usan cuando están soportadas por la base de datos subyacente. Con objeto de que se puedan usar tuplas de tamaño fijo (en la primera forma normal) para la representación de líneas poligonales se puede dar a la línea poligonal o a la curva un identificador, representando cada segmento como una tupla separada que también lleva el identificador de la línea poligonal o de la curva. De manera parecida, la representación triangulada de los polígonos permite una representación relacional de estos en su primera forma normal.

La representación de los puntos y de los segmentos rectilíneos en el espacio tridimensional es parecida a su representación en el espacio bidimensional, siendo la única diferencia que los puntos tienen un componente z adicional.

De manera parecida, la representación de las figuras planas, como los triángulos, los rectángulos y otros polígonos, no cambia mucho cuando se consideran tres dimensiones. Los tetraedros y los paralelepípedos pueden representarse de la misma manera que los triángulos y los rectángulos.

Es posible representar poliedros arbitrarios dividiéndolos en tetraedros, igual que se triangulan los polígonos. También se pueden representar indicando las caras, cada una de las cuales es, en sí misma, un polígono, junto con una indicación del lado de la cara que está por dentro del poliedro.

2 Algunas referencias usan el término *polígono cerrado* para hacer referencia a lo que aquí se denominan polígonos, y se refieren a las líneas poligonales abiertas como polígonos abiertos.

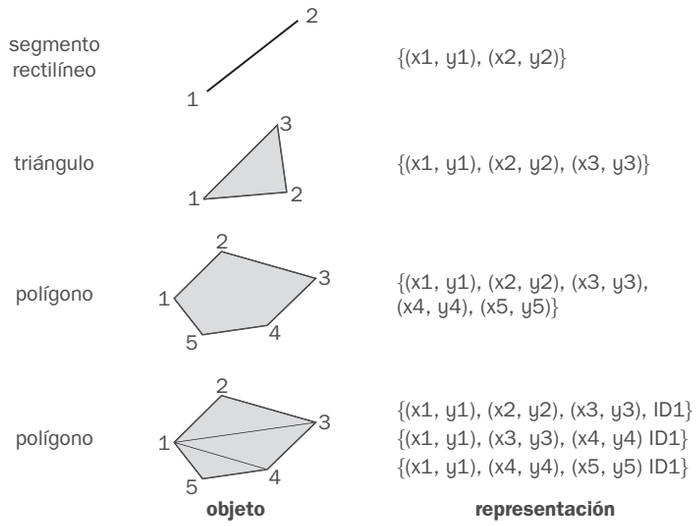


Figura 25.2. Representación de estructuras geométricas.

25.3.2. Bases de datos para diseño

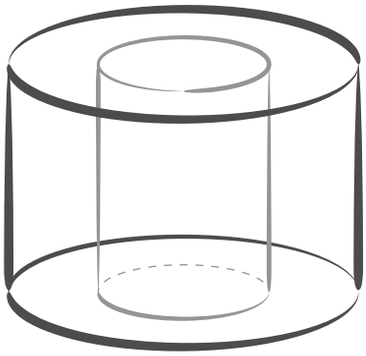
Los **sistemas de diseño asistido por computadora** (*computer aided design*: CAD) tradicionalmente almacenaban los datos en la memoria durante su edición u otro tipo de procesamiento y los volvían a escribir en archivos al final de la sesión de edición. Entre los inconvenientes de este esquema están el coste (la complejidad de programación y el coste temporal) de transformar los datos de una forma a otra, y la necesidad de leer todo un archivo aunque solo sea necesaria una parte. Para los diseños de gran tamaño, como el diseño de circuitos integrados a gran escala o el diseño de todo un avión, puede que resulte imposible guardar en la memoria el diseño completo. Los diseñadores de bases de datos orientadas a objetos estaban motivados en gran parte por las necesidades de los sistemas de CAD. Las bases de datos orientadas a objetos representan los componentes del diseño como objetos y las conexiones entre los objetos indican el modo en que está estructurado el diseño.

Los objetos almacenados en las bases de datos de diseño suelen ser objetos geométricos. Entre los objetos geométricos bidimensionales sencillos se encuentran los puntos, las líneas, los triángulos, los rectángulos y los polígonos en general. Los objetos bidimensionales complejos pueden formarse a partir de objetos sencillos mediante las operaciones de unión, intersección y diferencia. De manera parecida, los objetos tridimensionales complejos pueden formarse a partir de objetos más sencillos como las esferas, los cilindros y los paralelepípedos mediante las operaciones unión, intersección y diferencia, como en la Figura 25.3. Las superficies tridimensionales también pueden representarse mediante **modelos de alambres,** que esencialmente modelan las superficies como conjuntos de objetos más sencillos, como segmentos rectilíneos, triángulos y rectángulos.

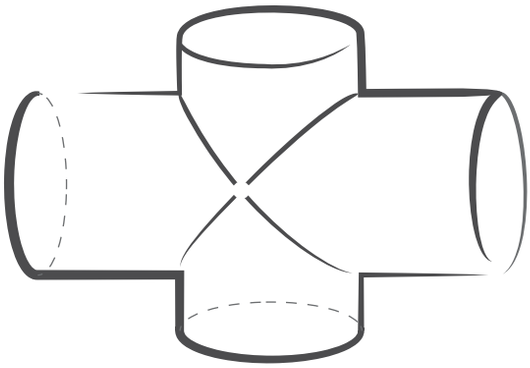
Las bases de datos de diseño también almacenan información no espacial sobre los objetos, como el material del que están construidos. Esa información se suele poder modelar mediante técnicas estándar de modelado de datos. Aquí se centrará la atención únicamente en los aspectos espaciales.

Al diseñar hay que realizar varias operaciones espaciales. Por ejemplo, puede que el diseñador desee recuperar la parte del diseño que corresponde a una región de interés determinada. Las estructuras espaciales de índices, estudiadas en la Sección 25.3.5, resultan útiles para estas tareas. Las estructuras espaciales de índices son multidimensionales, trabajan con datos de dos y de tres dimensiones, en vez de trabajar solamente con la sencilla ordenación unidimensional que proporcionan los árboles B^+.

Las restricciones de integridad espacial, como «dos tuberías no deben estar en la misma ubicación», son importantes en las bases de datos de diseño para evitar errores debidos a interferencias. Estos errores suelen producirse si el diseño se realiza a mano y solo se detectan al construir un prototipo. En consecuencia, estos errores pueden resultar costosos de corregir. El soporte de las bases de datos para las restricciones de integridad espacial ayuda a los usuarios a evitar errores de diseño, con lo que hacen que el diseño sea consistente. La implementación de esas verificaciones de integridad depende una vez más de la disponibilidad de estructuras multidimensionales de índices eficientes.



(a) Diferencia de cilindros



(b) Unión de cilindros

Figura 25.3. Objetos tridimensionales complejos.

25.3.3. Datos geográficos

Los datos geográficos son de naturaleza espacial, pero se diferencian de los datos de diseño en ciertos aspectos. Los mapas y las imágenes de satélite son ejemplos típicos de datos geográficos. Los mapas pueden proporcionar no solo información sobre la ubicación, sobre fronteras, ríos y carreteras, por ejemplo, sino también información mucho más detallada asociada con la ubicación, como la elevación, el tipo de suelo, el uso de la tierra y la cantidad anual de lluvia.

25.3.3.1. Aplicaciones de los datos geográficos

Las bases de datos geográficas tienen gran variedad de usos, incluidos los servicios de mapas en línea; los sistemas de navegación para vehículos; la información de redes de distribución para las empresas de servicios públicos, como son los sistemas de telefonía, electricidad y suministro de agua, y la información de uso de la tierra para ecologistas y planificadores.

Los servicios de mapas de carreteras basados en web constituyen una aplicación muy usada de datos cartográficos. En su nivel más sencillo, estos sistemas pueden usarse para generar mapas de carreteras en red de la región deseada. Una ventaja importante de los mapas interactivos es que resulta sencillo dimensionar los mapas al tamaño deseado, es decir, acercarse y alejarse para ubicar elementos importantes. Los servicios de mapas de carretera también almacenan información sobre carreteras y servicios, como el trazado de las carreteras, los límites de velocidad, las condiciones de las vías, las conexiones entre carreteras y los tramos de sentido único. Con esta información adicional sobre las carreteras se pueden usar los mapas para obtener indicaciones para desplazarse de un sitio a otro y para localizar, por ejemplo, hoteles, gasolineras o restaurantes con las ofertas y gamas de precios deseadas. En los últimos años, varios servicios de mapas en la web han definido API que permiten que los programadores creen mapas personalizados que incluyan datos de los servicios junto con datos de otras fuentes. Estos mapas personalizados se pueden utilizar para mostrar, por ejemplo, las casas en venta o alquiler, o las tiendas y restaurantes de una determinada área.

Los sistemas de navegación para los vehículos son sistemas montados en automóviles que proporcionan mapas de carreteras y servicios para la planificación de los viajes. Un añadido útil a los sistemas móviles de información geográfica, como los sistemas de navegación de los vehículos, es la unidad del **sistema de posicionamiento global** (*global positioning system*: GPS), que usa la información emitida por los satélites GPS para hallar la ubicación actual con una precisión de decenas de metros. Con estos sistemas, el conductor no puede perderse nunca;[3] la unidad GPS halla la ubicación en términos de latitud, longitud y elevación, y el sistema de navegación puede consultar la base de datos geográfica para hallar el lugar en que se encuentra y la carretera en que se halla el vehículo en el momento de la consulta.

Las bases de datos geográficas para información de utilidad pública se están volviendo cada vez más importantes a medida que crece la red de cables y tuberías enterrados. Sin mapas detallados las obras realizadas por una empresa de servicio público pueden dañar los cables de otra, lo que daría lugar a una interrupción del servicio a gran escala. Las bases de datos geográficas, junto con los sistemas precisos de determinación de la posición, pueden ayudar a evitar estos problemas.

25.3.3.2. Representación de los datos geográficos

Los **datos geográficos** pueden clasificarse en dos tipos:

- **Datos por líneas (raster).** Estos datos consisten en mapas de bits o en mapas de píxeles en dos o más dimensiones. Un ejemplo típico de imagen de líneas bidimensional son las imágenes de satélite de un área. Además de la imagen real, los datos incluyen la ubicación de esta, especificando por ejemplo la latitud y longitud de sus esquinas y la resolución, especificada por el número total de píxeles o, más común en el contexto de datos geográficos, por el área que cubre cada píxel.

  Los datos por líneas se suelen representar con **teselas;** cada una cubre un área de tamaño fijo. Se puede mostrar un área mayor mostrando todas las teselas que se superponen a esa área. Para permitir mostrar datos a diferentes niveles de ampliación, se crea un conjunto separado de teselas con cada nivel de ampliación. Cuando el usuario fija la ampliación con la interfaz, por ejemplo mediante un navegador de Internet, se obtienen y se muestran las teselas de ese nivel de ampliación, que se superponen con el área que se muestra.

  Los datos pueden ser tridimensionales, por ejemplo, la temperatura a diferentes altitudes en distintas regiones, también medida con la ayuda de un satélite. El tiempo puede formar otra dimensión, por ejemplo, las medidas de la temperatura de superficie en diferentes momentos.

3 Bueno, ¡casi nunca!

- **Datos vectoriales.** Los datos vectoriales están formados a partir de objetos geométricos básicos como los puntos, los segmentos rectilíneos, los triángulos y otros polígonos en dos dimensiones, y los cilindros, las esferas, los paralelepípedos y otros poliedros en tres dimensiones. En el contexto de los datos geográficos, los puntos se suelen representar por su latitud y longitud, y cuando la altura es relevante, adicionalmente por su elevación.

  Los datos cartográficos suelen representarse en formato vectorial. Las carreteras pueden representarse como uniones de varios segmentos rectilíneos. Elementos geográficos como los grandes lagos, o incluso los Estados y los países pueden representarse como polígonos. Algunos elementos como los ríos, se pueden representar como curvas complejas o como polígonos complejos, dependiendo de si su ancho es relevante o no.

La información geográfica relativa a las regiones, como la cantidad anual de lluvia, puede representarse como un array, es decir, en forma de líneas. Para reducir el espacio necesario, el array se puede almacenar de manera comprimida. En la Sección 25.3.5.2 se estudia una representación alternativa de estos arrays mediante una estructura de datos denominada *árbol cuadrático*.

Como alternativa, se puede representar la información regional en forma vectorial usando polígonos, cada uno de los cuales es una región en la que el valor del array es el mismo. En algunas aplicaciones la representación vectorial es más compacta que la de líneas. También es más precisa para algunas tareas, como los dibujos de carreteras, en los que la división de la región en píxeles (que pueden ser bastante grandes) lleva a una pérdida de precisión en la información de la ubicación. No obstante, la representación vectorial resulta inadecuada para las aplicaciones en las que los datos se basan en líneas de manera intrínseca, como las imágenes de satélite.

La información **topográfica,** es decir, información sobre la elevación (altura) de cada punto sobre la superficie, se puede representar en forma de líneas (raster). Alternativamente se puede representar en forma vectorial dividiendo la superficie en polígonos que cubran las regiones de (aproximadamente) la misma elevación, con un único valor de elevación asociado a cada uno de los polígonos. Otra alternativa es que la superficie se **triangule;** es decir, se divida en triángulos, donde cada triángulo se representa con la latitud y longitud de cada uno de sus vértices. Esta última representación se denomina **red irregularmente triangulada (TIN);** es una representación compacta especialmente útil para generar vistas tridimensionales de un área.

Los sistemas de información geográfica normalmente contienen tanto datos de líneas como vectoriales y pueden mezclar ambos tipos de datos cuando muestran los resultados al usuario. Por ejemplo, las aplicaciones de mapas normalmente contienen tanto imágenes de satélite como datos vectoriales de carreteras, edificios y otros elementos. En la representación de un mapa normalmente se **superponen** diferentes tipos de información; por ejemplo, la información de carreteras se puede superponer sobre una imagen de satélite para crear una visualización híbrida. De hecho, un mapa normalmente consta de varias capas, que se muestran de la inferior a la superior; los datos de las capas superiores aparecen sobre los datos de las capas inferiores.

También es interesante hacer notar que aunque la información se almacene en forma vectorial se puede convertir a raster antes de ser enviada a una interfaz de usuario como, por ejemplo, un navegador web. Una razón es que incluso los navegadores que no disponen de lenguajes de guiones (necesarios para interpretar y mostrar datos vectoriales) puedan mostrar datos cartográficos; una segunda razón puede ser evitar que los usuarios extraigan y utilicen los datos vectoriales.

Los servicios cartográficos como Google Maps y Yahoo! Maps proporcionan API para que los usuarios puedan crear mapas especializados, con datos de aplicación específicos que se muestran superpuestos sobre los mapas estándar. Por ejemplo, un sitio web puede mostrar un área con información de restaurantes sobre el mapa. Esta superposición se puede crear dinámicamente, mostrando solamente los restaurantes de un tipo de cocina, o permitiendo que los usuarios cambien el nivel de ampliación o desplacen el mapa. Las API de mapas para un determinado lenguaje (normalmente JavaScript o Flash) se construyen sobre servicios web que proporcionan los datos cartográficos subyacentes.

25.3.4. Consultas espaciales

Existen varios tipos de consultas con referencia a ubicaciones espaciales.

- Las **consultas de proximidad** solicitan objetos que se encuentren cerca de una ubicación especificada. La consulta para encontrar todos los restaurantes a menos de una distancia dada de un determinado punto es un ejemplo de consulta de proximidad. La **consulta de vecino más próximo** solicita el objeto que se halla más próximo al punto especificado. Por ejemplo, puede que se desee encontrar la gasolinera más cercana. Observe que esta consulta no tiene que especificar un límite para la distancia y, por tanto, se puede formular aunque no se tenga idea de la distancia a la que se halla la gasolinera más próxima.
- Las **consultas regionales** tratan de regiones espaciales. Estas consultas pueden preguntar por objetos que se hallen parcial o totalmente en el interior de la región especificada. Un ejemplo es la consulta para encontrar todas las tiendas minoristas dentro de los límites geográficos de una ciudad dada.
- Puede que las consultas también soliciten **intersecciones** y **uniones** de regiones. Por ejemplo, dada la información regional, como pueden ser la lluvia anual y la densidad de población, una consulta puede solicitar todas las regiones con una baja cantidad de lluvia anual y una elevada densidad de población.

Las consultas que calculan las intersecciones de regiones pueden considerarse como si calcularan la **reunión espacial** de dos relaciones espaciales, por ejemplo, una que represente la cantidad de lluvia y otra que represente la densidad de población, siendo la ubicación el atributo de reunión. En general, dadas dos relaciones, cada una de las cuales contiene objetos espaciales, la reunión espacial de las dos relaciones genera, o bien pares de objetos que se intersectan, o bien las regiones de intersección de esos pares.

Existen diferentes algoritmos de reunión que calculan eficazmente las reuniones espaciales de datos vectoriales. Aunque se pueden usar las reuniones de bucles anidados o de bucles anidados indexados (con índices espaciales), las reuniones de asociación y las reuniones por mezcla-ordenación no se pueden usar con datos espaciales. Los investigadores han propuesto técnicas de reunión basadas en el recorrido coordinado de las estructuras espaciales de los índices de las dos relaciones. Consulte las notas bibliográficas para obtener más información.

En general, las consultas de datos espaciales pueden tener una combinación de requisitos espaciales y no espaciales. Por ejemplo, puede que se desee averiguar el restaurante más cercano que tenga menú vegetariano y que cueste menos de diez euros por comida.

Dado que los datos espaciales son inherentemente gráficos, se suelen consultar mediante un lenguaje gráfico de consulta. El resultado de esas consultas también se muestra gráficamente, en vez de mostrarse en tablas. El usuario puede realizar varias operaciones con la interfaz, como escoger el área que desea ver (por ejemplo, apuntando y pulsando en los barrios del oeste de Manhattan), acercarse y alejarse, escoger lo que desea mostrar de acuerdo con las condiciones de selección (por ejemplo, casas con más de tres ha-

bitaciones), superponer varios mapas (por ejemplo, las casas con más de tres habitaciones superpuestas sobre un mapa que muestre las zonas con bajas tasas de delincuencia), etc. La interfaz gráfica constituye la parte visible para el usuario. Se han propuesto extensiones de SQL para permitir que las bases de datos relacionales almacenen y recuperen información espacial de manera eficiente y que las consultas mezclen las condiciones espaciales con las no espaciales. Las extensiones incluyen la autorización de tipos de datos abstractos como las líneas, los polígonos y los mapas de bits, y permiten condiciones espaciales como *contiene* o *superpone*.

25.3.5. Índices sobre los datos espaciales

Los índices son necesarios para el acceso eficiente a los datos espaciales. Las estructuras de índices tradicionales, como los índices de asociación y los árboles B, no resultan adecuadas, ya que únicamente trabajan con datos unidimensionales, mientras que los datos espaciales suelen ser de dos o más dimensiones.

25.3.5.1. Los árboles k-d

Para comprender el modo de indexar los datos espaciales que constan de dos o más dimensiones se considera en primer lugar el indexado de los puntos de los datos unidimensionales. Las estructuras arbóreas, como los árboles binarios y los árboles B, operan dividiendo el espacio en partes más pequeñas de manera sucesiva. Por ejemplo, cada nodo interno de un árbol binario divide un intervalo unidimensional en dos. Los puntos que quedan en la partición izquierda van al subárbol izquierdo; los puntos que quedan en la partición de la derecha van al subárbol derecho. En los árboles binarios equilibrados la partición se escoge de modo que, aproximadamente, la mitad de los puntos almacenados en el subárbol caigan en cada partición. De manera parecida, cada nivel de un árbol B divide un intervalo unidimensional en varias partes.

Se puede usar esa intuición para crear estructuras arbóreas para el espacio bidimensional, así como para espacios de más dimensiones. Una estructura arbórea denominada **árbol k-d** fue una de las primeras estructuras usadas para la indexación en varias dimensiones. Cada nivel de un árbol k-d divide el espacio en dos. La división se realiza según una dimensión en el nodo del nivel superior del árbol, según otra dimensión en los nodos del nivel siguiente, etc., alternando cíclicamente las dimensiones. La división se realiza de tal modo que, en cada nodo, aproximadamente la mitad de los puntos almacenados en el subárbol cae a un lado y la otra mitad al otro. La división se detiene cuando un nodo tiene menos puntos que un valor máximo dado. La Figura 25.4 muestra un conjunto de puntos en el espacio bidimensional y una representación en árbol k-d de ese conjunto de puntos. Cada línea corresponde a un nodo del árbol, y el número máximo de puntos en cada nodo hoja se ha definido como uno. Cada línea de la figura (aparte del marco exterior) corresponde a un nodo del árbol k-d. La numeración de las líneas en la figura indica el nivel del árbol en el que aparece el nodo correspondiente.

El **árbol k-d-B** extiende el árbol k-d para permitir varios nodos hijo por cada nodo interno, igual que los árboles B extienden los árboles binarios, para reducir la altura del árbol. Los árboles k-d-B están mejor adaptados para el almacenamiento secundario que los árboles k-d.

25.3.5.2. Árboles cuadráticos

Una representación alternativa de los datos bidimensionales son los **árboles cuadráticos.** En la Figura 25.5 aparece un ejemplo de la división del espacio mediante un árbol cuadrático. El conjunto de puntos es el mismo que en la Figura 25.4. Cada nodo de un árbol cuadrático está asociado con una región rectangular del espacio. El nodo superior está asociado con todo el espacio objetivo. Cada nodo que no sea un nodo hoja del árbol cuadrático divide su región en cuatro cuadrantes del mismo tamaño y, a su vez, cada uno de esos nodos tiene cuatro nodos hijo correspondientes a los cuatro cuadrantes. Los nodos hoja tienen un número de puntos que varía entre cero y un número máximo fijado. A su vez, si la región correspondiente a un nodo tiene más puntos que el máximo fijado, se crean nodos hijo para ese nodo. En el ejemplo de la Figura 25.5, el número máximo de puntos de cada nodo hoja está fijado en uno.

Este tipo de árbol cuadrático se denomina **árbol cuadrático PR,** para indicar que almacena los puntos y que la división del espacio se basa en regiones, en vez de en el conjunto real de puntos almacenados. Se pueden usar **árboles cuadráticos regionales** para almacenar información de arrays (de líneas). Cada nodo de los árboles cuadráticos regionales es un nodo hoja si todos los valores del array de la región que abarca son iguales. En caso contrario, se vuelve a subdividir en cuatro nodos hijo con la misma área y es, por tanto, un nodo interno. Cada nodo del árbol cuadrático regional corresponde a un subarray de valores. Los subarrays correspondientes a las hojas contienen un solo elemento del array o varios, todos ellos con el mismo valor.

El indexado de los segmentos rectilíneos y de los polígonos presenta problemas nuevos. Existen extensiones de los árboles k-d y de los árboles cuadráticos para esta labor. No obstante, un segmento rectilíneo o un polígono pueden cruzar una línea divisoria. Si lo hacen, hay que dividirlos y representarlo en cada uno de los subárboles en que aparezcan sus fragmentos. La aparición múltiple de un segmento lineal o de un polígono puede dar lugar a ineficiencias en el almacenamiento, así como a ineficiencias en las consultas.

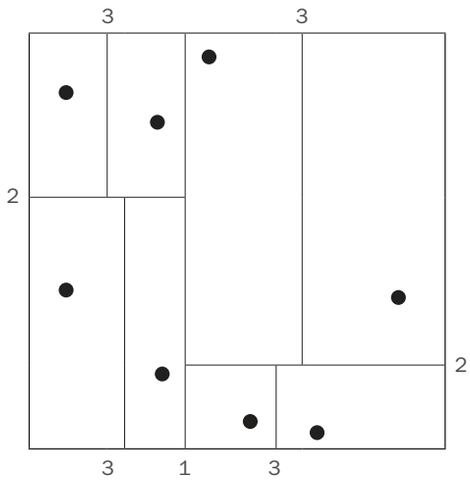


Figura 25.4. División del espacio por un árbol k-d.

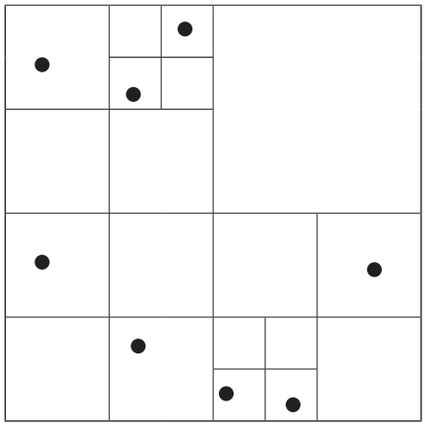


Figura 25.5. División del espacio por un árbol cuadrático.

25.3.5.3. Árboles R

La estructura de almacenamiento denominada **árbol R** resulta útil para el indexado de objetos como puntos, segmentos de línea, rectángulos y otros polígonos. Un árbol R es una estructura arbórea equilibrada con los objetos indexados almacenados en los nodos hoja, de manera parecida a los árboles B+. No obstante, en lugar de un rango de valores, con cada nodo del árbol se asocia una **caja límite** rectangular. La caja límite de un nodo hoja es el rectángulo mínimo paralelo a los ejes que contiene todos los objetos almacenados en el nodo hoja. La caja límite de los nodos internos, de manera parecida, es el rectángulo mínimo paralelo a los ejes que contiene las cajas límite de sus nodos hijo. La caja límite de un objeto (como un polígono) viene definida, de manera parecida, como el rectángulo mínimo paralelo a los ejes que contiene al objeto.

Cada nodo interno almacena las cajas límite de los nodos hijo junto con los punteros para los nodos hijo. Cada nodo hijo almacena los objetos indexados y puede que, opcionalmente, almacene las cajas límite de esos objetos; las cajas límite ayudan a acelerar las comprobaciones de solapamientos de los rectángulos con los objetos indexados; si el rectángulo de una consulta no se solapa con la caja límite de un objeto, no se puede solapar tampoco con el objeto. (Si los objetos indexados son rectángulos, por supuesto no hace falta almacenar las cajas límite, ya que son idénticas a los rectángulos).

La Figura 25.6 muestra el ejemplo de un conjunto de rectángulos (dibujados con línea continua) y de sus cajas límite (dibujadas con línea discontinua) de los nodos de un árbol R para el conjunto de rectángulos. Hay que tener en cuenta que las cajas límite se muestran con espacio adicional en su interior, para hacerlas visibles en el dibujo. En realidad, las cajas son menores y se ajustan fielmente a los objetos que contienen; es decir, cada lado de una caja límite *C* toca como mínimo uno de los objetos o de las cajas límite que se contienen en *C*.

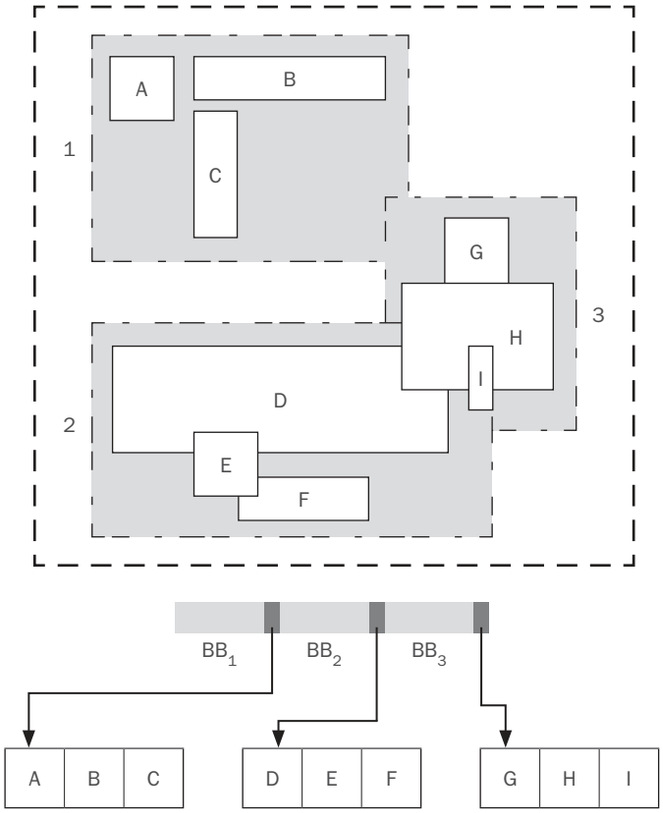


Figura 25.6. Árbol R.

El árbol R en sí mismo se halla a la derecha de la Figura 25.6. La figura hace referencia a las coordenadas de la caja límite *i* como CL_i.

Ahora se verá el modo de implementar las operaciones de búsqueda, inserción y eliminación en los árboles R.

- **Búsqueda.** Como muestra la figura, las cajas límite con nodos hermanos pueden solaparse; en los árboles B+, los árboles k-d y los árboles cuadráticos, sin embargo, los rangos no se solapan. La búsqueda de objetos que contengan un punto, por tanto, tiene que seguir *todos* los nodos hijo cuyas cajas límites asociadas contengan el punto; en consecuencia, puede que haya que buscar por varios caminos. De manera parecida, una consulta para buscar todos los objetos que intersecten con uno dado tiene que bajar por todos los nodos en los que el rectángulo asociado intersecte al objeto dado.
- **Inserción.** Cuando se inserta un objeto en un árbol R se selecciona un nodo hoja para que lo guarde. Lo ideal sería seleccionar un nodo hoja que tuviera espacio para guardar una nueva entrada, y cuya caja límite contuviera a la caja límite del objeto. Sin embargo, puede que ese nodo no exista; aunque exista, hallarlo puede resultar muy costoso, ya que no es posible encontrarlo mediante un único recorrido desde la raíz. En cada nodo interno se pueden encontrar varios nodos hijo cuyas cajas límite contengan la caja límite del objeto, y hay que explorar cada uno de esos nodos hijo. Por tanto, como norma heurística, en un recorrido desde la raíz, si alguno de los nodos hijo tiene una caja límite que contenga a la caja límite del objeto, el algoritmo del árbol R escoge una de ellas de manera arbitraria. Si ninguno de los nodos hijo satisface esta condición, el algoritmo escoge un nodo hijo cuya caja límite tenga el solapamiento máximo con la caja límite del objeto para continuar el recorrido.

  Una vez que se ha llegado al nodo hoja, si este ya está lleno, el algoritmo lleva a cabo una división del nodo (y propaga hacia arriba la división si fuese necesario) de manera muy parecida a la inserción de los árboles B+. Igual que con la inserción en los árboles B+, el algoritmo de inserción de los árboles R asegura que el árbol siga equilibrado. Además, asegura que las cajas límite de los nodos hoja, así como los nodos internos, sigan siendo consistentes; es decir, las cajas límite de los nodos hoja contienen todas las cajas límite de los polígonos almacenados en el nodo hoja, mientras que las cajas límite de los nodos internos contienen todas las cajas límite de los nodos hijo.

  La principal diferencia entre el procedimiento de inserción y la inserción en los árboles B+ radica en el modo en que se dividen los nodos. En los árboles B+ es posible hallar un valor tal que la mitad de los de las entradas sea menor que el punto medio y la mitad sea mayor que el valor. Esta propiedad no se generaliza más allá de una dimensión; es decir, para más de una dimensión, no siempre es posible dividir las entradas en dos conjuntos tales que sus cajas límite no se superpongan. En su lugar, como norma heurística, el conjunto de entradas *S* puede dividirse en dos conjuntos disjuntos S_1 y S_2 tales que las cajas límite de S_1 y S_2 tengan un área total mínima; otra norma heurística sería dividir las entradas en dos conjuntos S_1 y S_2 de modo que S_1 y S_2 tengan un solapamiento mínimo. Los dos nodos resultantes de la división contendrían las entradas de S_1 y S_2, respectivamente. El coste de hallar las divisiones con área total o solapamiento mínimo puede ser elevado, por lo que se usan normas heurísticas más económicas, como la heurística de la *división cuadrática* (la heurística recibe el nombre del hecho de que toma el cuadrado del tiempo en el número de entradas).

  La heurística de la **división cuadrática** funciona de esta manera: en primer lugar, selecciona un par de entradas *a* y *b* de *S* tales que al ponerlas en el mismo nodo den lugar a una caja límite con el máximo de espacio desaprovechado; es decir, el área de la caja límite mínima de *a* y *b* menos la suma de las áreas de *a* y *b* es máxima. La heurística sitúa las entradas *a* y *b* en los conjuntos S_1 y S_2, respectivamente.

Luego añade iterativamente las entradas restantes, una por cada iteración, a uno de los dos conjuntos S_1 o S_2. En cada iteración, para cada entrada restante e, $i_{e,1}$ denota el incremento en el tamaño de la caja límite de S_1 si e se añade a S_1 e $i_{e,2}$ denota el incremento correspondiente de S_2. En cada iteración, la heurística escoge una de las entradas con la máxima diferencia entre $i_{e,1}$ e $i_{e,2}$ y la añade a S_1 si $i_{e,1}$ es menor que $i_{e,2}$, y a S_2 en caso contrario. Es decir, se escoge en cada iteración una entrada con «preferencia máxima» por S_1 o S_2. Las iteraciones se detienen cuando se han asignado todas las entradas o cuando uno de los conjuntos S_1 o S_2 tiene entradas suficientes como para que todas las demás entradas haya que añadirlas al otro conjunto, de modo que los nodos creados a partir de S_1 y S_2 tengan la ocupación mínima exigida. La heurística añade luego todas las entradas no asignadas al conjunto con menos entradas.

- **Eliminación.** La eliminación puede llevarse a cabo como si fuera una eliminación de árbol B^+, tomando prestadas las entradas de los nodos hermanos, o mezclando nodos hermanos si un nodo se queda menos lleno de lo exigido. Un enfoque alternativo redistribuye todas las entradas de los nodos menos llenos de lo necesario a los nodos hermanos, con el objetivo de mejorar el agrupamiento de las entradas en el árbol R.

Consulte las referencias bibliográficas para obtener más detalles sobre las operaciones de inserción y eliminación en los árboles R, así como sobre las variantes de los árboles R, denominados árboles R^* o árboles R^+.

La eficiencia del almacenamiento de los árboles R es mayor que la de los árboles k-d y que la de los árboles cuadráticos, ya que cada polígono solo se almacena una vez, y se puede asegurar que cada nodo está, como mínimo, medio lleno. No obstante, las consultas pueden resultar más lentas, ya que hay que buscar por varios caminos. Las reuniones espaciales son más sencillas con los árboles cuadráticos que con los árboles R, ya que todos los árboles cuadráticos de cada región están divididos de la misma manera. Sin embargo, debido a su mayor eficiencia de almacenamiento y a su parecido con los árboles B, los árboles R y sus variantes se han hecho populares en los sistemas de bases de datos que soportan datos espaciales.

25.4. Bases de datos multimedia

Los datos multimedia, como las imágenes, el sonido y el vídeo, una modalidad de datos cada vez más popular, en la actualidad casi siempre se almacenan fuera de las bases de datos, en sistemas de archivos. Este tipo de almacenamiento no supone ningún problema cuando el número de objetos multimedia es relativamente pequeño, ya que las características proporcionadas por las bases de datos no suelen ser importantes.

Sin embargo, las características de las bases de datos se vuelven importantes cuando el número de objetos multimedia almacenados es grande. Aspectos como las actualizaciones transaccionales, las facilidades de consulta y el indexado se vuelven importantes. Los objetos multimedia suelen tener atributos descriptivos, como los que indican su fecha de creación, su creador y la categoría a la que pertenecen. Un enfoque de la creación de una base de datos para esos objetos multimedia es usar las bases de datos para almacenar los atributos descriptivos y realizar un seguimiento de los archivos en los que se almacenan los objetos multimedia.

Sin embargo, el almacenamiento de los objetos multimedia fuera de la base de datos hace más difícil proporcionar funciones de base de datos, como el indexado con base en el contenido real de datos multimedia. También puede provocar inconsistencias, como que un archivo esté registrado en la base de datos pero sus contenidos falten, o viceversa. Por tanto, resulta deseable almacenar los propios datos en la base de datos.

Si se pretende almacenar los datos multimedia en una base de datos hay que abordar varios aspectos:

- La base de datos debe soportar objetos de gran tamaño, ya que los datos multimedia como los vídeos pueden ocupar varios gigabytes de espacio de almacenamiento. Los objetos de mayor tamaño pueden dividirse en fragmentos menores. De manera alternativa, los objetos multimedia pueden almacenarse en un sistema de archivos, pero la base de datos puede contener un puntero hacia el objeto; el puntero suele ser un nombre de archivo. El estándar SQL/MED (MED significa Management of External Data, gestión de datos externos) permite que los datos externos, como los archivos, se traten como si formaran parte de la base de datos. Con SQL/MED parece que los objetos son parte de la base de datos, pero pueden almacenarse externamente. Los formatos de los datos multimedia se estudian en la Sección 25.4.1.
- La recuperación de algunos tipos de datos, como los de sonido y los de vídeo, exige que la entrega de los datos deba realizarse a una velocidad constante garantizada. Estos datos se denominan a veces **datos isócronos** o **datos de medios continuos.** Por ejemplo, si los datos de sonido no se proporcionan a tiempo, habrá saltos en el sonido. Si los datos se proporcionan demasiado deprisa, se pueden desbordar las memorias intermedias, lo que dará lugar a la pérdida de datos. Los datos de medios continuos se estudian en la Sección 25.4.2.
- La recuperación basada en la semejanza es necesaria en muchas aplicaciones de bases de datos multimedia. Por ejemplo, en una base de datos que almacene imágenes de huellas dactilares, se proporciona una consulta de la imagen de una huella dactilar y hay que recuperar las huellas dactilares de la base de datos que sean parecidas a la huella dactilar de la consulta. Las estructuras de índices como los árboles B^+ y los árboles R no se pueden usar para esta finalidad; hay que crear estructuras especiales de índices. La recuperación basada en el parecido se estudia en la Sección 25.4.3.

25.4.1. Formatos de datos multimedia

Debido al gran número de bytes necesarios para representar los datos multimedia es fundamental que se almacenen y transmitan de manera comprimida. Para los datos de imágenes, el formato más usado es *JPEG*, que recibe su nombre del organismo de normalización que lo creó: el grupo conjunto de expertos en imágenes (*Joint Picture Experts Group*). Los datos de vídeo se pueden almacenar codificando cada fotograma de vídeo en formato JPEG, pero esa codificación supone un desperdicio, ya que los fotogramas de vídeo sucesivos suelen ser casi iguales. El grupo de expertos en películas (*Moving Picture Experts Group*) desarrolló la serie de estándares *MPEG* para codificar los datos de vídeo y de audio; estas codificaciones aprovechan las similitudes de las secuencias de fotogramas para conseguir un grado de compresión mayor. El estándar *MPEG-1* almacena un minuto de vídeo y sonido a 30 fotogramas por segundo en unos 12,5 megabytes (en comparación con los aproximadamente 75 megabytes solo para vídeo en JPEG). No obstante, la codificación MPEG-1 introduce alguna pérdida de calidad del vídeo, a un nivel aproximadamente equivalente al de las cintas de vídeo VHS. El estándar *MPEG-2* se diseñó para los sistemas de radiodifusión digitales y para los discos de vídeo digitales (DVD); solo introduce una pérdida de calidad de vídeo despreciable. MPEG-2 comprime un minuto de vídeo y de sonido en aproximadamente 17 megabytes. MPEG-4 proporciona técnicas de mayor compresión de vídeo con ancho de banda variable para proporcionar la difusión de vídeo en redes con un amplio rango de anchos de banda. Para la codificación de sonido se usan varios estándares competidores, entre ellos *MP3* (que significa MPEG-1 Capa 3), RealAudio, Windows Media Audio y otros formatos. El vídeo en alta definición con el audio se codifica con distintas variantes de *MPEG-4*, que incluyen MPEG-4 AVC y AVCHD.

25.4.2. Datos de medios continuos

Los tipos de datos de medios continuos más importantes son los datos de vídeo y los de audio (por ejemplo, una base de datos de películas). Los sistemas de medios continuos se caracterizan por sus requisitos de entrega de información en tiempo real:

- Los datos deben entregarse lo bastante rápido como para que no haya saltos en la entrega del audio ni del vídeo.
- Los datos deben entregarse a una velocidad que no cause un desbordamiento de las memorias intermedias del sistema.
- La sincronización entre los distintos flujos de datos debe conservarse. Esta necesidad surge, por ejemplo, cuando el vídeo de una persona que habla muestra sus labios moviéndose de manera sincronizada con el sonido de su voz.

Para proporcionar los datos de manera predecible en el momento correcto a un gran número de consumidores debe coordinarse cuidadosamente la captura de estos desde el disco. Generalmente, los datos se capturan en ciclos periódicos. En cada ciclo, digamos de n segundos, se capturan n segundos de datos para cada consumidor y se almacenan en las memorias intermedias, mientras los datos capturados en el ciclo anterior se envían a los consumidores desde las memorias intermedias. El periodo de los ciclos es un compromiso: un periodo corto usa menos memoria pero necesita más movimientos del brazo del disco, lo que supone un desperdicio de recursos, mientras que un periodo largo reduce el movimiento del brazo del disco pero aumenta las necesidades de memoria y puede retrasar la entrega inicial de datos. Cuando llega una nueva solicitud entra en acción el **control de admisión:** es decir, el sistema comprueba si se puede satisfacer la solicitud con los recursos disponibles (en cada periodo); si es así, se admite; en caso contrario, se rechaza.

La extensa investigación realizada sobre la entrega de datos de medios continuos ha tratado aspectos como el manejo de arrays de discos y el tratamiento de los fallos de los discos. Consulte las referencias bibliográficas para obtener más detalles.

Varios fabricantes ofrecen servidores de vídeo bajo demanda. Los sistemas actuales están basados en los sistemas de archivos, ya que los sistemas de bases de datos existentes no proporcionan la respuesta en tiempo real que necesitan estas aplicaciones. La arquitectura básica de un sistema de vídeo bajo demanda consta de:

- **Servidor de vídeo.** Los datos multimedia se almacenan en varios discos (generalmente en una configuración RAID). Los sistemas que contienen un gran volumen de datos puede que utilicen medios de almacenamiento terciario para los datos a los que se tiene acceso con menor frecuencia.
- **Terminales.** Las personas visualizan los datos multimedia mediante distintos dispositivos denominados *terminales*. Ejemplos de estos son las computadoras personales y los televisores conectados a una computadora pequeña y de coste reducido denominada **decodificador (set-top box).**
- **Red.** La transmisión de los datos multimedia desde el servidor hasta los terminales necesita una red de gran capacidad.

Actualmente, el servicio de vídeo bajo demanda en redes de cable está disponible en muchos lugares y acabará siendo ubicuo, igual que lo son ahora la televisión por ondas hercianas y la televisión por cable.

25.4.3. Recuperación basada en la semejanza

En muchas aplicaciones multimedia los datos solo se describen en la base de datos de manera aproximada. Un ejemplo son los datos de huellas dactilares comentados en la Sección 25.4.

Otros ejemplos son:

- **Datos gráficos.** Dos gráficos o imágenes que sean ligeramente diferentes en su representación en la base de datos pueden ser considerados iguales por un usuario. Por ejemplo, una base de datos puede almacenar diseños de marcas comerciales. Cuando haya que registrar una nueva marca puede que el sistema necesite identificar antes todas las marcas parecidas que se registraron anteriormente.
- **Datos de sonido.** Se están desarrollando interfaces de usuario basadas en el reconocimiento de la voz que permiten a los usuarios dar un comando o identificar un elemento de datos por la voz. Debe comprobarse la semejanza de la entrada del usuario con los comandos o los elementos de datos almacenados en el sistema.
- **Datos manuscritos.** La entrada manuscrita puede usarse para identificar un elemento de datos manuscritos o una orden manuscrita almacenados en la base de datos. Una vez más, se necesita comprobar la semejanza.

El concepto de semejanza suele ser subjetivo y específico del usuario. Sin embargo, la verificación de la semejanza suele tener más éxito que el reconocimiento de voz o de letras manuscritas, ya que la entrada puede compararse con datos que ya se encuentran en el sistema y, por tanto, el conjunto de opciones disponibles para el sistema es limitado.

Existen varios algoritmos para encontrar las mejores coincidencias con una entrada dada mediante la comprobación de la semejanza. Algunos sistemas, incluido un sistema telefónico de llamada por nombre activado por la voz, se han distribuido comercialmente. Consulte las notas bibliográficas para más referencias.

25.5. Movilidad y bases de datos personales

Las bases de datos comerciales de gran tamaño se han almacenado tradicionalmente en instalaciones informáticas centrales. En las aplicaciones de bases de datos distribuidas generalmente ha habido una fuerte administración central de las bases de datos y de la red. Se han combinado las siguientes tendencias tecnológicas para crear aplicaciones en las cuales la suposición de una administración y de un control centralizados no es completamente correcta:

- El uso cada vez más extendido de computadoras personales y, sobre todo, de computadoras portátiles.
- El uso cada vez más extendido de teléfonos móviles con capacidades de una computadora.
- El desarrollo de una infraestructura inalámbrica de comunicaciones digitales de coste relativamente bajo, basada en redes inalámbricas de área local, redes celulares de paquetes digitales y otras tecnologías.

La **computación móvil** se ha mostrado útil en gran cantidad de aplicaciones. Muchos profesionales usan computadoras portátiles para poder trabajar y tener acceso a los datos durante sus viajes. Los servicios de mensajería usan computadoras portátiles para ayudar al seguimiento de los paquetes. Los servicios de emergencia usan computadoras portátiles en el escenario de los desastres, en las emergencias médicas y similares para tener acceso a la información e introducir datos relativos a la situación. Los teléfonos móviles (celulares) se están convirtiendo en dispositivos que no solo proporcionan servicios de telefonía, sino también computación móvil, que permite disponer de correo electrónico y acceso web. Siguen surgiendo nuevas aplicaciones de las computadoras portátiles.

Las computadoras con comunicación inalámbrica generan una situación en la que las máquinas ya no tienen ubicaciones ni direcciones de red fijas. Las **consultas dependientes de la ubicación** son una clase interesante de consultas que están motivadas por las computadoras portátiles; en estas consultas la ubicación del usuario (computadora) es un parámetro de la consulta. El valor del parámetro de ubicación lo proporciona el usuario o, cada vez más, un sistema de posicionamiento global (GPS). Un ejemplo son los sistemas de información para viajeros que proporcionan a los conductores datos sobre los hoteles, los servicios de carretera y similares. El procesamiento de las consultas sobre los servicios que se hallan más adelante en la ruta actual debe basarse en la ubicación del usuario, en su dirección de movimiento y en su velocidad. Cada vez se ofrecen más ayudas a la navegación como una característica integrada en los automóviles.

La energía (la carga de las baterías) es un recurso escaso para la mayor parte de las computadoras portátiles. Esta limitación influye en muchos aspectos del diseño de los sistemas. Entre las consecuencias más interesantes de la necesidad de eficiencia energética es que los pequeños dispositivos móviles emplean la mayor parte del tiempo hibernándose, despertándose durante una fracción de segundo cada segundo, aproximadamente, para comprobar si hay datos entrantes y para enviar datos. Este comportamiento tiene un impacto significativo en los protocolos usados para comunicarse con los dispositivos móviles. El uso de emisiones programadas de datos en los sistemas móviles para disminuir la necesidad de transmisión de consultas es otra forma de reducir los requisitos de energía.

Cantidades cada vez mayores de datos residen en máquinas administradas por los usuarios en lugar de por administradores de bases de datos. Además, estas máquinas pueden estar, a veces, desconectadas de la red. En muchos casos hay un conflicto entre la necesidad del usuario de seguir trabajando mientras está desconectado y la necesidad de consistencia global de los datos.

Es probable que un usuario utilice más de un dispositivo móvil. Estos usuarios necesitan ser capaces de ver sus datos en su versión más reciente independientemente de qué dispositivo estén usando en cada momento. Normalmente, esta capacidad es posible gracias a alguna variante de computación en la nube, ya tratadas en la Sección 19.9.

En las Secciones 25.5.1 a 25.5.4 se estudian técnicas en uso y en desarrollo para tratar los problemas de las computadoras portátiles y de la informática personal.

25.5.1. Un modelo de computación móvil

El entorno de la computación móvil consiste en computadoras portátiles, denominadas **anfitriones móviles,** y una red de computadoras conectadas por cable. Los anfitriones móviles se comunican con la red de cable mediante computadoras denominadas **estaciones para el soporte de movilidad.** Cada estación para el soporte de movilidad gestiona los anfitriones móviles de su **celda;** es decir, del área geográfica que cubre. Los anfitriones móviles pueden moverse de unas celdas a otras, por lo que necesitan el **relevo** del control de una estación para el soporte de movilidad a otra. Dado que los anfitriones móviles pueden, a veces, estar apagados, un anfitrión puede abandonar una celda y aparecer más tarde en otra distante. Por tanto, los movimientos de unas celdas a otras no se realizan necesariamente entre celdas adyacentes. Dentro de un área pequeña, como un edificio, los anfitriones móviles pueden conectarse mediante una red inalámbrica de área local (LAN), que proporciona conectividad de coste más reducido que las redes celulares de área amplia, y que reduce la sobrecarga de los relevos.

Es posible que los anfitriones móviles se comuniquen directamente sin intervención de ninguna estación para el soporte de movilidad. No obstante, esa comunicación solo puede ocurrir entre anfitriones cercanos. Estas formas directas de comunicación se están haciendo más frecuentes con la llegada del estándar **Bluetooth,** una norma de radio digital de corto alcance para permitir la conectividad por radio a alta velocidad (hasta 721 kilobits por segundo) a distancias de alrededor de diez metros. Concebido inicialmente como una sustitución de los cables, lo más prometedor del Bluetooth es la conexión ad hoc sencilla entre computadoras portátiles, PDA, teléfonos celulares y los denominados dispositivos inteligentes.

Los sistemas de redes de área local inalámbricas basados en las normas 801.11 (a/b/g/n) se usan mucho actualmente y se están desplegando los sistemas basados en 802.16 (Wi-Max).

La infraestructura de red para la computación móvil se basa en gran parte en dos tecnologías: redes de área local inalámbricas y redes de telefonía celular basadas en paquetes. Los primeros sistemas celulares usaban tecnología analógica y estaban diseñados para la comunicación de voz. Los sistemas digitales de segunda generación siguieron centrándose en las aplicaciones de voz. Los sistemas de tercera generación (3G) y los denominados sistemas 2.5G usan redes basadas en paquetes y están más adaptados a las aplicaciones de datos. En estas redes, la voz es solo una más de las aplicaciones (aunque una económicamente importante). Las tecnologías de cuarta generación (4G) incluyen Wi-Max y otros competidores.

Bluetooth, las redes de área local inalámbricas y las redes celulares 2.5G y 3G hacen posible que se comuniquen a bajo coste gran variedad de dispositivos. Aunque esta comunicación en sí misma no encaja en el dominio de las aplicaciones habituales de bases de datos, los datos de la contabilidad, del control y de la administración correspondientes a esta comunicación generan bases de datos enormes. La inmediatez de la comunicación inalámbrica provoca la necesidad de acceso en tiempo real a muchas de estas bases de datos. Esta necesidad de inmediatez añade otra dimensión a las restricciones del sistema, un asunto que se abordará en profundidad en la Sección 26.4.

El tamaño y las limitaciones de potencia de muchas computadoras portátiles han llevado a la creación de jerarquías de memoria alternativas. En lugar de, o además de, el almacenamiento en disco, puede incluirse la memoria flash, que se estudió en la Sección 10.1. Si el anfitrión móvil incluye un disco duro, puede que se permita que el disco deje de girar cuando no se utilice, para ahorrar energía. Las mismas consideraciones de tamaño y de energía limitan el tipo y el tamaño de las pantallas usadas en los dispositivos portátiles. Los diseñadores de los dispositivos portátiles suelen crear interfaces de usuario especiales para trabajar con estas restricciones. No obstante, la necesidad de presentar datos basados en web ha exigido la creación de estándares para presentaciones. El **protocolo de aplicaciones inalámbrico** (*wireless application protocol*: WAP) es un estándar para el acceso inalámbrico a Internet. Los exploradores basados en WAP tienen acceso a páginas web especiales que usan el **lenguaje de marcas inalámbrico** (*wireless markup language*: WML), un lenguaje basado en XML diseñado para las restricciones de la exploración web móvil e inalámbrica.

25.5.2. Dirección y procesamiento de consultas

La ruta entre cada par de anfitriones puede cambiar con el tiempo si alguno de los dos anfitriones es móvil. Este sencillo hecho tiene un efecto espectacular en la red, ya que las direcciones de red basadas en las ubicaciones ya no son constantes en el sistema.

La computación móvil también afecta directamente al procesamiento de consultas de las bases de datos. Como se vio en el Capítulo 19, hay que considerar los costes de comunicación cuando se escoge una estrategia de procesamiento distribuido de las consultas. La informática móvil hace que los costes de comunicación cambien de manera dinámica, lo que complica el proceso de optimización.

Además, hay varios conceptos de coste que hay que considerar:

- El **tiempo del usuario** es un elemento muy valioso en muchas aplicaciones profesionales.

- El **tiempo de conexión** es la unidad por la que se asignan los costes monetarios en algunos sistemas de telefonía celular.
- El **número de bytes, o de paquetes, transferidos** es la unidad por la que se calculan los costes en algunos sistemas de telefonía celular digital.
- Los **costes basados en la hora del día** varían en función de si la comunicación se produce durante los periodos pico o durante los periodos valle.
- La **energía** es limitada. A menudo la carga de las baterías es un recurso escaso cuyo uso debe optimizarse. Un principio básico de las comunicaciones inalámbricas es que hace falta menos energía para recibir señales de radio que para emitirlas. Así, la transmisión y la recepción de los datos imponen demandas de energía diferentes al anfitrión móvil.

25.5.3. Datos de difusión

Suele ser deseable para los datos que se solicitan con frecuencia que los transmitan estaciones de soporte de las computadoras portátiles en un ciclo continuo, en lugar de que se transmitan a los anfitriones móviles a petición de estos. Una aplicación típica de estos **datos de difusión** es la información de las cotizaciones bursátiles. Hay dos motivos para usar los datos de difusión. En primer lugar, los anfitriones móviles evitan el coste energético de transmitir las solicitudes de datos. En segundo lugar, los datos de difusión pueden recibirlos simultáneamente gran número de anfitriones móviles, sin coste adicional. Por tanto, el ancho de banda disponible para las transmisiones se usa de manera más efectiva.

Así, los anfitriones móviles pueden recibir los datos a medida que se transmiten, en lugar de consumir energía transmitiendo solicitudes. Puede que los anfitriones móviles tengan almacenamiento no volátil disponible para guardar en la caché los datos de difusión para su uso posterior. Dada una consulta, los anfitriones móviles pueden minimizar los costes energéticos determinando si pueden procesarla solo con los datos guardados en la caché. Si los datos guardados en la caché son insuficientes, hay dos opciones: esperar a que los datos se transmitan o transmitir una solicitud de datos. Para tomar esta decisión, los anfitriones móviles deben conocer el momento en que se transmitirán los datos en cuestión.

Los datos de difusión pueden transmitirse de acuerdo con una programación fija o según una programación variable. En el primer caso, los anfitriones móviles usan la programación fija conocida para determinar el momento en que se transmitirán los datos en cuestión. En el segundo caso, se debe transmitir la propia programación de transmisiones en una frecuencia de radio conocida y a intervalos de tiempos también conocidos.

En efecto, el medio transmitido puede modelarse como un disco de gran latencia. Las solicitudes de datos pueden considerarse atendidas cuando se transmiten los datos solicitados. Las programaciones de transmisión se comportan como índices de los discos. En las notas bibliográficas aparecen trabajos de investigación recientes en el área de administración de los datos de difusión.

25.5.4. Desconexiones y consistencia

Dado que puede que las comunicaciones inalámbricas se paguen con arreglo al tiempo de conexión, hay un incentivo para que determinados anfitriones móviles se desconecten durante periodos de tiempo considerables. Las computadoras portátiles sin conectividad inalámbrica están desconectadas la mayor parte del tiempo en que se usan, excepto cuando se conectan de manera periódica a sus computadoras anfitrionas, físicamente o mediante una red.

Durante esos periodos de desconexión puede que el anfitrión móvil siga operativo. Es posible que el usuario del anfitrión móvil formule consultas y solicite actualizaciones de los datos que residen localmente o que se han guardado en la caché local. Esta situación crea varios problemas, en especial:

- **Recuperabilidad.** Las actualizaciones introducidas en una máquina desconectada pueden perderse si el anfitrión móvil sufre un fallo catastrófico. Dado que el anfitrión móvil representa un único punto de fallo, no se puede simular bien el almacenamiento estable.
- **Consistencia.** Los datos guardados en la caché local pueden quedar obsoletos, pero el anfitrión móvil no puede descubrir la situación hasta que vuelva a conectarse. De manera parecida, las actualizaciones que se generen en el anfitrión móvil no pueden transmitirse hasta que no se produzca de nuevo la conexión.

El problema de la consistencia se estudió en el Capítulo 19, en el que se vieron las particiones de la red, y aquí se partirá de esa base. En los sistemas distribuidos conectados por redes físicas las particiones se consideran un modo de fallo; en la computación móvil las particiones mediante desconexiones son parte del modo de operación normal. Por tanto, es necesario permitir que continúe el acceso a los datos a pesar de las particiones, pese al riesgo de que se produzca una pérdida de consistencia.

Para los datos actualizados solo por el anfitrión móvil, es sencillo transmitir las actualizaciones cuando el anfitrión móvil vuelve a conectarse. No obstante, si el anfitrión móvil guarda en la caché copias de los datos solo para lectura que pueden actualizar otras computadoras, es posible que los datos guardados en la caché acaben siendo inconsistentes. Cuando se conecta el anfitrión móvil, puede recibir **informes de invalidación** que informen de que las entradas de la caché están obsoletas. No obstante, cuando el anfitrión móvil esté desconectado puede perder algún informe de invalidación. Una solución sencilla a este problema es invalidar toda la caché al volver a conectar el anfitrión móvil, pero una solución tan extrema resulta muy costosa. En las notas bibliográficas se citan varios esquemas para el almacenamiento en la caché.

Si se pueden producir actualizaciones tanto en el anfitrión móvil como en el resto del sistema, la detección de las actualizaciones conflictivas resulta más difícil. Los esquemas basados en la **numeración de versiones** permiten la actualización de los archivos compartidos desde los anfitriones desconectados. Estos esquemas no garantizan que las actualizaciones sean consistentes. Más bien garantizan que si dos anfitriones actualizan de manera independiente la misma versión del documento, el conflicto se acabará descubriendo cuando los anfitriones intercambien información directamente o mediante un anfitrión común.

El **esquema del vector de versiones** detecta las inconsistencias cuando las copias de un documento se actualizan de manera independiente. Este esquema permite que las copias de un *documento* se almacenen en varios anfitriones. Aunque se utilice el término *documento*, el esquema puede aplicarse a otros elementos de datos, como las tuplas de una relación.

La idea básica es que cada anfitrión i almacene, con su copia de cada documento d, un **vector de versiones,** es decir, un conjunto de números de versiones $\{V_{d,i}[j]\}$, con una entrada para cada uno de los demás anfitriones d en los que se puede actualizar potencialmente el documento. Cuando un anfitrión i actualiza un documento d, incrementa el número de versión $V_{d,i}[i]$ en una unidad.

Siempre que dos anfitriones i y j se conectan entre sí, intercambian los documentos actualizados, de modo que los dos obtienen versiones nuevas de los documentos. No obstante, antes de intercambiar los documentos los anfitriones tienen que averiguar si las copias son consistentes:

1. Si los vectores de versiones son iguales en los dos anfitriones, es decir, si para cada k, $V_{d,i}[k] = V_{d,j}[k]$, las copias del documento d son idénticas.

2. Si para cada k, $V_{d,i}[k] \leq V_{d,j}[k]$ y los vectores de versiones no son idénticos, la copia del documento d del anfitrión i es más antigua que la del anfitrión j. Es decir, la copia del documento d del anfitrión j se obtuvo mediante una o más modificaciones de la copia del documento del anfitrión i. El anfitrión i sustituye su copia de d, así como su copia del vector de versiones de d, por las copias del anfitrión j.
3. Si hay un par de anfitriones k y m tales que $V_{d,i}[k] < V_{d,j}[k]$ y $V_{d,i}[m] > V_{d,j}[m]$, las copias son *inconsistentes*; es decir, la copia de d de i contiene actualizaciones realizadas por el anfitrión k que no se han transmitido al anfitrión j y, de manera parecida, la copia de d de j contiene actualizaciones llevadas a cabo por el anfitrión m que no se han transmitido al anfitrión i. Entonces, las copias de d son inconsistentes, ya que se han realizado de manera independiente dos o más actualizaciones de d. Puede que se necesite la intervención manual para mezclar las actualizaciones.

El esquema de vectores de versiones se diseñó inicialmente para tratar los fallos en los sistemas de archivos distribuidos. El esquema adquirió mayor importancia porque las computadoras portátiles suelen almacenar copias de los archivos que también se hallan presentes en los sistemas servidores, lo que constituye, de facto, un sistema de archivos distribuido que suele estar desconectado. Otra aplicación de este esquema es el de sistemas de trabajo en grupo, en los que los anfitriones se conectan de manera periódica, en lugar de hacerlo de manera continua, y deben intercambiar los documentos actualizados.

El esquema del vector de versiones también tiene aplicaciones en las bases de datos replicadas, donde se puede aplicar a tuplas individuales. Por ejemplo, si se mantiene un calendario o libreta de direcciones en un dispositivo móvil y en un anfitrión, las inserciones, borrados y actualizaciones pueden producirse tanto en el dispositivo móvil como en el anfitrión. Aplicando el esquema de vector de versiones a las entradas individuales del calendario o los contactos resulta sencillo manejar las situaciones en las que se ha actualizado una determinada entrada en el dispositivo móvil y otra diferente en el anfitrión; esta situación no se considera un conflicto. Sin embardo, si la misma entrada se actualiza de forma independiente en ambas partes se detectaría un conflicto gracias al esquema de vector de versiones.

No obstante, el esquema del vector de versiones no logra abordar el problema más difícil e importante que plantean las actualizaciones de los datos compartidos: la reconciliación de las copias inconsistentes de los datos. Muchas aplicaciones pueden llevar a cabo de manera automática la reconciliación ejecutando en cada computadora las operaciones que han conducido a las actualizaciones en las computadoras remotas durante el periodo de desconexión. Esta solución funciona si las operaciones de actualización son conmutativas; es decir, generan el mismo resultado, independientemente del orden en que se ejecuten. Puede que se disponga de técnicas alternativas en ciertas aplicaciones; en el peor de los casos, no obstante, debe dejarse a los usuarios que resuelvan las inconsistencias. El tratamiento automático de estas inconsistencias y la ayuda a los usuarios para que resuelvan las que no puedan tratarse de manera automática sigue siendo un área actual de investigación.

Otra debilidad es que el esquema del vector de versiones exige una comunicación sustancial entre el anfitrión móvil que vuelve a conectarse y su estación para el soporte de movilidad. Las verificaciones de la consistencia pueden posponerse hasta que se necesiten los datos, aunque este retraso puede incrementar la inconsistencia global de la base de datos.

La posibilidad de desconexión y el coste de las comunicaciones inalámbricas limitan el aspecto práctico de las técnicas de procesamiento de las transacciones para los sistemas distribuidos estudiadas en el Capítulo 19. A menudo resulta preferible dejar que los usuarios preparen las transacciones en los anfitriones móviles y exigir que, en lugar de ejecutarlas localmente, las remitan al servidor para su ejecución. Las transacciones que afectan a más de una computadora y que incluyen un anfitrión móvil afrontan bloqueos de larga duración durante el compromiso de la transacción, a menos que las desconexiones sean raras o predecibles.

25.6. Resumen

- El tiempo desempeña un papel importante en los sistemas de bases de datos. Las bases de datos son modelos del mundo real. Aunque la mayor parte de las bases de datos modelan el estado del mundo real en un momento dado (en el momento actual), las bases de datos temporales modelan los estados del mundo real a lo largo del tiempo.
- Los hechos de las relaciones temporales tienen momentos asociados cuando son válidos, que pueden representarse como una unión de intervalos. Los lenguajes de consultas temporales simplifican el modelado del tiempo, así como las consultas relacionadas con el tiempo.
- Las bases de datos espaciales se usan cada vez más hoy en día para almacenar datos de diseño asistido por computadora y datos geográficos.
- Los datos de diseño se almacenan sobre todo como datos vectoriales; los datos geográficos consisten en una combinación de datos vectoriales y lineales (raster). Las restricciones de integridad espacial son importantes para los datos de diseño.
- Los datos vectoriales pueden codificarse como datos de la primera forma normal o almacenarse mediante estructuras que no sean la primera forma normal, como las listas. Las estructuras de índices de finalidad espacial resultan especialmente importantes para tener acceso a los datos espaciales y para procesar las consultas espaciales.
- Los árboles R son una extensión multidimensional de los árboles B; con variantes como los árboles R^+ y los árboles R^*, se han hecho populares en las bases de datos espaciales. Las estructuras de índices que dividen el espacio de manera regular, como los árboles cuadráticos, ayudan a procesar las consultas de mezcla espaciales.
- Las bases de datos multimedia están aumentando en importancia. Problemas como la recuperación basada en la semejanza y la entrega de datos a velocidades garantizadas son temas de investigación actuales.
- Los sistemas de computación móvil se han vuelto de uso común, lo que ha despertado el interés por los sistemas de bases de datos que pueden ejecutarse en ellos. El procesamiento de las consultas en estos sistemas puede implicar la búsqueda en las bases de datos de los servidores. El modelo de coste de las consultas debe incluir el coste de la comunicación, incluido el coste monetario y el coste de la energía de las baterías, que resulta relativamente elevado para los sistemas portátiles.
- La transmisión por receptor resulta mucho más económica que la comunicación punto a punto, y la difusión de datos como los datos bursátiles ayuda a los sistemas portátiles a recoger los datos de manera económica.
- La operación en desconexión, el uso de los datos de difusión y el almacenamiento de los datos en la caché son tres problemas importantes que se están abordando hoy en día en la informática móvil.

Términos de repaso

- Datos temporales.
- Tiempo válido.
- Tiempo de transacción.
- Relación temporal.
- Relación bitemporal.
- Tiempo universal coordinado (UTC).
- Relación instantánea.
- Lenguajes de consultas temporales.
- Selección temporal.
- Proyección temporal.
- Reunión temporal.
- Datos espaciales y geográficos.
- Datos de diseño asistido por computadora (*computer aided design*: CAD).
- Datos geográficos.
- Sistemas de información geográfica.
- Triangulación.
- Bases de datos de diseño.
- Datos geográficos.
- Datos por líneas (raster).
- Datos vectoriales.
- Sistema de posicionamiento global (*global positioning system*: GPS).
- Consultas espaciales.
- Consultas de proximidad.
- Consultas de vecino más próximo.
- Consultas regionales.
- Reunión espacial.
- Indexado de los datos espaciales.
- Árboles k-d.
- Árboles k-d-B.
- Árboles cuadráticos.
  - PR.
  - Regional.
- Árboles R.
  - Caja límite.
  - División cuadrática.
- Bases de datos multimedia.
- Datos isócronos.
- Datos de medios continuos.
- Recuperación basada en la semejanza.
- Formatos de datos multimedia.
- Servidores de vídeo.
- Computación móvil.
  - Anfitriones móviles.
  - Estaciones de soporte de movilidad.
  - Celda.
  - Relevo.
- Consultas dependientes de la ubicación.
- Datos de difusión.
- Consistencia.
  - Informes de invalidación.
  - Esquema del vector de versiones.

Ejercicios prácticos

25.1. Indique los dos tipos de tiempo y en qué se diferencian. Explique el motivo de que haya dos tipos de tiempo asociados con cada tupla.

25.2. Suponga que se tiene una relación que contiene las coordenadas x e y y nombres de restaurantes. Suponga también que las únicas consultas que se formularán serán de la siguiente forma: la consulta especifica un punto y pregunta si hay algún restaurante exactamente en ese punto. Indique el tipo de índice que sería preferible, árbol R o árbol B. Justifique la respuesta.

25.3. Suponga que se dispone de una base de datos espacial que soporta consultas regionales (con regiones circulares) pero no consultas de vecino más próximo. Describa un algoritmo para hallar el vecino más próximo haciendo uso de varias consultas regionales.

25.4. Suponga que se desean almacenar segmentos rectilíneos en un árbol R. Si un segmento rectilíneo no es paralelo a los ejes, su caja límite puede ser grande y contener un área vacía grande.

- Describa el efecto en el rendimiento de tener cajas límite de gran tamaño en las consultas que piden los segmentos rectilíneos que intersectan una región dada.
- Describa brevemente una técnica para mejorar el rendimiento de esas consultas y describa un ejemplo de sus ventajas. (Sugerencia: se pueden dividir los segmentos en partes más pequeñas).

25.5. Indique un procedimiento recursivo para calcular de manera eficiente la mezcla espacial de dos relaciones con índices de árbol R. (Sugerencia: use cajas límite para comprobar si las entradas hoja bajo un par de nodos internos se pueden intersectar).

25.6. Describa el modo en que las ideas subyacentes a la organización RAID (Sección 10.3) se pueden usar en un entorno de datos de difusión, en el que puede que ocasionalmente haya ruido que impida la recepción de parte de los datos que se están transmitiendo.

25.7. Defina un modelo en el que se difundan repetidamente los datos modelando el medio de transmisión como un disco virtual. Describa el modo en que el tiempo de acceso y la velocidad de transferencia de datos del disco virtual se diferencian de los valores correspondientes a un disco duro normal.

25.8. Considere una base de datos de documentos en la que todos los documentos se conserven en una base de datos central. En las computadoras portátiles se guardan copias de algunos documentos. Suponga que la computadora portátil A actualiza una copia del documento 1 mientras está desconectada y que, al mismo tiempo, la computadora portátil B actualiza una copia del documento 2 mientras está desconectada. Muestre el modo en que el esquema del vector de versiones puede asegurar la actualización adecuada de la base de datos central y de las computadoras portátiles cuando se vuelva a conectar una computadora portátil.

Ejercicios

25.9. Indique si se conservarán las dependencias funcionales si se convierte una relación en una relación temporal añadiendo un atributo temporal. Indique el modo en que se resuelve el problema en las bases de datos temporales.

25.10. Considere dos datos vectoriales bidimensionales en los que los elementos de datos no se solapan. Indique si es posible convertir esos datos vectoriales en datos lineales (raster). En caso de que sea posible, indique los inconvenientes de almacenar los datos lineales obtenidos de esa conversión en lugar de los datos vectoriales originales.

25.11. Estudie el soporte de los datos espaciales ofrecido por el sistema de bases de datos que se está usando e implemente lo siguiente:

a. Un esquema para representar la ubicación geográfica de los restaurantes y sus características, como el tipo de cocina que se sirve en cada restaurante y su nivel de precios.

b. Una consulta para encontrar los restaurantes más económicos que sirven comida india y que estén a menos de nueve kilómetros de la casa del lector (suponga cualquier ubicación para la casa del lector).

c. Una consulta para encontrar para cada restaurante su distancia al restaurante más cercano que sirve la misma cocina y con el mismo nivel de precios.

25.12. Indique los problemas que se producen en un sistema de medios continuos si los datos se entregan demasiado lento o demasiado rápido.

25.13. Indique tres características principales de la computación móvil en redes inalámbricas que son diferentes de las de los sistemas distribuidos tradicionales.

25.14. Indique tres factores que haya que considerar en la optimización de las consultas para la computación móvil que no se consideren en los optimizadores de consultas tradicionales.

25.15. Describa un ejemplo para mostrar que el esquema del vector de versiones no asegura la secuenciabilidad. (Sugerencia: utilice el ejemplo del Ejercicio práctico 25.8, con la suposición de que los documentos 1 y 2 están disponibles en las dos computadoras portátiles A y B, teniendo en cuenta la posibilidad de que un documento pueda leerse sin que esté actualizado).

Notas bibliográficas

Stam y Snodgrass [1988] y Soo [1991] proporcionan estudios sobre la administración de los datos temporales. Jensen et ál. [1994] presentan un glosario de conceptos de las bases de datos temporales, con la intención de unificar la terminología. Tansel et ál. [1993] es una colección de artículos sobre diferentes aspectos de las bases de datos temporales. Chomicki [1995] presenta técnicas para administrar las restricciones para la integridad temporal.

Heywood et ál. [2002] es un libro que estudia los sistemas de información geográfica. Samet [1995b] proporciona una introducción a la gran cantidad de trabajo realizado sobre las estructuras espaciales de índices. Samet [1990] y Samet [2006] proporcionan una cobertura en el nivel de los libros de texto de las estructuras espaciales de datos. Una de las primeras descripciones de los árboles cuadráticos se proporciona en Finkel y Bentley [1974]. Samet [1990] y Samet [1995b] describen numerosas variantes de los árboles cuadráticos. Bentley [1975] describe los árboles k-d, y Robinson [1981] describe los árboles k-d B. Los árboles R se presentaron originalmente en Guttman [1984]. Las extensiones de los árboles R se presentan en Sellis et ál. [1987], que describen los árboles R^+, y Beckmann et ál. [1990], que describen los árboles R^*.

Brinkhoff et ál. [1993] estudian una implementación de las mezclas espaciales mediante árboles R. Lo y Ravishankar [1996], y Patel y DeWitt [1996] presentan los métodos basados en las particiones para el cálculo de las reuniones espaciales. Samet y Aref [1995] proporcionan una introducción de los modelos espaciales de datos, de las operaciones espaciales y de la integración de los datos espaciales con los no espaciales.

Revesz [2002] es un libro que estudia el área de las bases de datos con restricciones; los intervalos temporales y las regiones espaciales se pueden considerar un caso especial de restricciones.

Samet [1995a] describe los campos de investigación en las bases de datos multimedia. El indexado de los datos multimedia se estudia en Faloutsos y Lin [1995].

Dashti et ál. [2003] es un libro que describe el diseño de servidores de flujos de datos de medios, incluyendo un estudio extensivo de la organización de datos en subsistemas de discos. Los servidores de vídeo se estudian en Anderson et ál. [1992], Rangan et ál. [1992], Ozden et ál. [1994], Freedman y DeWitt [1995], y Ozden et ál. [1996b]. La tolerancia a los fallos se estudia en Berson et ál. [1995] y Ozden et ál. [1996a].

La administración de la información en los sistemas que incluyen computadoras portátiles se estudia en Alonso y Korth [1993] y en Imielinski y Badrinath [1994]. Imielinski y Korth [1996] presentan una introducción a la computación móvil y una colección de trabajos de investigación sobre el tema.

El esquema del vector de versiones para la detección de la inconsistencia en los sistemas de archivos distribuidos se describe en Popek et ál. [1981] y en Parker et ál. [1983].

Procesamiento avanzado de transacciones

26

En los Capítulos 14, 15 y 16 se introdujo el concepto de transacción, que es una unidad de programa que tiene acceso, y posiblemente actualiza, varios elementos de datos, y cuya ejecución asegura la conservación de las propiedades ACID. En esos capítulos se estudiaron gran variedad de esquemas para asegurar las propiedades ACID en entornos en los que pueden producirse fallos, y en los que las transacciones pueden ejecutarse de manera concurrente.

En este capítulo se irá más allá de los esquemas básicos estudiados anteriormente y se abordarán los conceptos del procesamiento avanzado de las transacciones, incluidos los monitores de procesamiento de transacciones, los flujos de trabajo de las transacciones y el procesamiento de transacciones en el contexto del comercio electrónico. También se estudian las bases de datos en memoria principal, las bases de datos en tiempo real, las transacciones de larga duración y las transacciones anidadas.

26.1. Monitores de procesamiento de transacciones

Los **monitores de procesamiento de transacciones** (monitores TP) son sistemas que se desarrollaron en los años setenta y ochenta del siglo pasado, inicialmente como respuesta a la necesidad de soportar gran número de terminales remotos (como los terminales de reserva de las líneas aéreas) desde una sola computadora. El término *monitor TP* significaba inicialmente *monitor de teleprocesamiento*.

Los monitores TP han evolucionado desde entonces para ofrecer el soporte central para el procesamiento distribuido de las transacciones, y el término monitor TP ha adquirido su significado actual. El monitor TP CICS de IBM fue uno de los primeros monitores TP, y se ha utilizado mucho. Otros monitores son Tuxedo de Oracle y Microsoft Transaction Server.

Las arquitecturas de servidores de aplicaciones web, incluyendo los servlets, que se estudiaron anteriormente en la Sección 9.3, proporcionan soporte a muchas de las características de los monitores TP y a veces se conocen como «TP ligeros». Los servidores de aplicaciones web se usan ampliamente y han reemplazado a los monitores TP tradicionales para muchas aplicaciones. Sin embargo, los conceptos subyacentes en ellos, que se estudian en este apartado, son esencialmente iguales.

26.1.1. Arquitecturas de los monitores TP

Los sistemas de procesamiento de transacciones a gran escala se construyen en torno a una arquitectura cliente-servidor. Una manera de crear estos sistemas es tener un proceso servidor para cada cliente; el servidor realiza la autentificación y luego ejecuta las acciones solicitadas por el cliente. Este **modelo de proceso por cliente** se ilustra en la Figura 26.1a. Este modelo presenta varios problemas con respecto al uso de la memoria y a la velocidad de procesamiento:

- Los requisitos de memoria para cada proceso son elevados. Aunque se comparta la memoria para el código de los programas entre todos los procesos, cada proceso consume memoria para los datos locales y los descriptores de los archivos abiertos, así como para la sobrecarga del sistema operativo, como las tablas de páginas para soportar la memoria virtual.
- El sistema operativo divide el tiempo disponible de CPU entre los procesos conmutando entre ellos; esta tarea se denomina **multitarea.** Cada **cambio de contexto** entre un proceso y el siguiente supone una sobrecarga considerable de la CPU; incluso en los rápidos sistemas de hoy en día, un cambio de contexto puede tardar cientos de microsegundos.

Los problemas anteriores se pueden evitar mediante un proceso con un solo servidor al que se conecten todos los clientes remotos; este modelo se denomina **modelo de servidor único,** ilustrado en la Figura 26.1b. Los clientes remotos envían las solicitudes al proceso del servidor, que ejecuta entonces esas solicitudes. Este modelo también se usa en los entornos cliente-servidor, en los que los clientes envían solicitudes a un proceso de un solo servidor. El proceso servidor asume las tareas, como la autentificación de los usuarios, que normalmente asumiría el sistema operativo. Para evitar bloquear a otros clientes al procesar una solicitud de larga duración de un cliente, el servidor usa **multienhebramiento:** el proceso servidor crea una hebra de control para cada cliente y, en efecto, implementa su propia multitarea de baja sobrecarga. Ejecuta el código en nombre de un cliente durante un rato, luego guarda el contexto interno y conmuta al código de otro cliente. A diferencia de la sobrecarga que supone la multitarea, el coste de la conmutación entre hebras es reducido (generalmente solo unos pocos microsegundos).

Los sistemas basados en el modelo de servidor único, como la versión original del monitor TP CICS de IBM y los servidores de archivos como NetWare de Novell, proporcionaban con éxito tasas de transacciones elevadas con recursos limitados. No obstante, presentaban problemas, especialmente cuando varias aplicaciones tenían acceso a la misma base de datos:

- Dado que todas las aplicaciones se ejecutan como un único proceso, no hay protección entre ellas. Un fallo en una aplicación puede afectar a todas las aplicaciones. Sería mejor ejecutar cada aplicación como un proceso separado.
- Estos sistemas no resultan adecuados para las bases de datos paralelas o distribuidas, ya que un proceso servidor no puede ejecutarse simultáneamente en varios servidores. (Sin embargo, las hebras concurrentes de un proceso pueden soportarse en un sistema multiprocesador de memoria compartida). Se trata de un inconveniente serio para las organizaciones de gran tamaño en las que el procesamiento paralelo resulta fundamental para el tratamiento de grandes cargas de trabajo, y los datos distribuidos son cada vez más frecuentes.

Una manera de resolver estos problemas es ejecutar varios procesos del servidor de aplicaciones que tengan acceso a una base de datos común y dejar que los clientes se comuniquen con la aplicación mediante un único proceso de comunicaciones que encamine las solicitudes. Este modelo se denomina **modelo de varios servidores y un solo encaminador (router),** que se muestra en la Figura 26.1c. Este modelo soporta procesos de servidor independientes para varias aplicaciones; además, cada aplicación puede tener un grupo de procesos de servidor, cualquiera de los cuales puede manejar una sesión cliente. La solicitud puede, por ejemplo, encaminarse al servidor con menor carga de un grupo. Como antes, cada proceso de servidor puede tener, a su vez, varias hebras, de modo que puede atender de manera concurrente a varios clientes. Como generalización adicional, los servidores de aplicaciones pueden ejecutarse en sitios diferentes de una base de datos paralela o distribuida y el proceso de comunicaciones puede manejar las comunicaciones entre los procesos.

La arquitectura anterior también se usa mucho en los servidores web. Un servidor web tiene un proceso principal que recibe las solicitudes HTTP, y luego asigna la tarea de manejar cada solicitud a un proceso diferente (escogido de entre un grupo de procesos). Cada uno de los procesos tiene, a su vez, varias hebras, por lo que puede atender varias solicitudes. El uso de lenguajes de programación seguros, tales como Java, C#, o Visual Basic, permite que los servidores de aplicaciones web protejan las hebras frente a errores producidos en otras. En cambio, con un lenguaje como C o C++ los errores como los de asignación de memoria en una hebra pueden provocar el fallo de otras.

Una arquitectura más general tiene varios procesos, en lugar de uno solo, para comunicarse con los clientes. Los procesos de comunicación con los clientes interactúan con uno o varios procesos encaminadores, que encaminan las solicitudes hacia el servidor correspondiente. Los monitores TP de generaciones posteriores, por tanto, tienen una arquitectura diferente, denominada **modelo de varios servidores y varios encaminadores,** que se muestra en la Figura 26.1d. Un proceso controlador inicia los demás procesos y supervisa su funcionamiento. Los sistemas servidores web de rendimiento muy elevado también adoptan una arquitectura de este tipo. Los procesos del encaminador son generalmente encaminadores de red que dirigen el tráfico de la misma dirección Internet a distintas computadoras servidoras, dependiendo de dónde venga el tráfico. Lo que parece al mundo exterior un solo servidor con una sola dirección puede ser una colección de servidores.

La estructura detallada de un monitor TP aparece en la Figura 26.2. Un monitor TP hace más cosas que pasar mensajes a los servidores de aplicaciones. Cuando llegan los mensajes, puede que haya que ubicarlos en una cola; por tanto, hay un **gestor de colas** para los mensajes entrantes. Puede que la cola sea una **cola duradera,** cuyas entradas sobreviven a los fallos del sistema. El uso de colas duraderas ayuda a asegurar que se acaben procesando los mensajes una vez recibidos y guardados en la cola, independientemente de los fallos del sistema. La gestión de las autorizaciones y de los servidores de aplicaciones (por ejemplo, el inicio de los servidores y el encaminamiento de los mensajes hacia los servidores) son otras funciones de los monitores TP. Los monitores TP suelen proporcionar recursos para la elaboración de registros históricos y la recuperación y el control de concurrencia, lo que permite a los servidores de aplicaciones implementar directamente, si fuera necesario, las propiedades ACID de las transacciones.

Finalmente, los monitores TP también proporcionan soporte a la mensajería persistente. Hay que recordar que la mensajería persistente (Sección 19.4.3) proporciona una garantía de que el mensaje se entregue si (y solo si) la transacción se compromete.

Además de estos servicios, muchos monitores TP también proporcionaban *recursos para presentaciones* con el fin de crear interfaces de menús o formularios para los clientes no inteligentes, como los terminales; estos recursos ya no son importantes porque ya no se usan mucho los clientes no inteligentes.

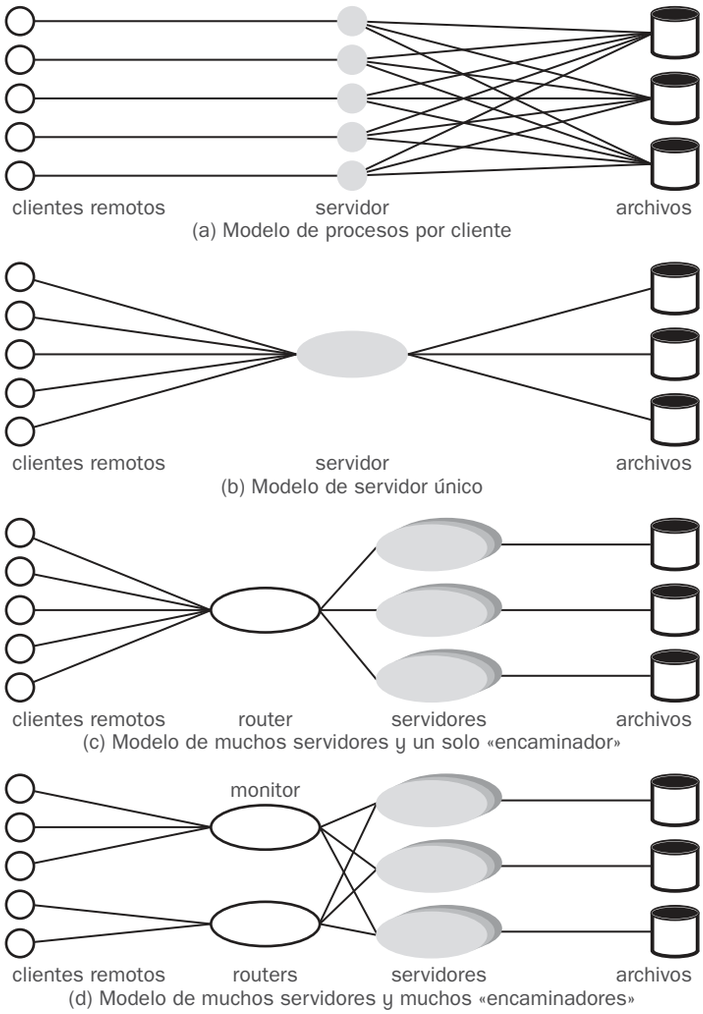


Figura 26.1. Arquitecturas de los monitores TP.

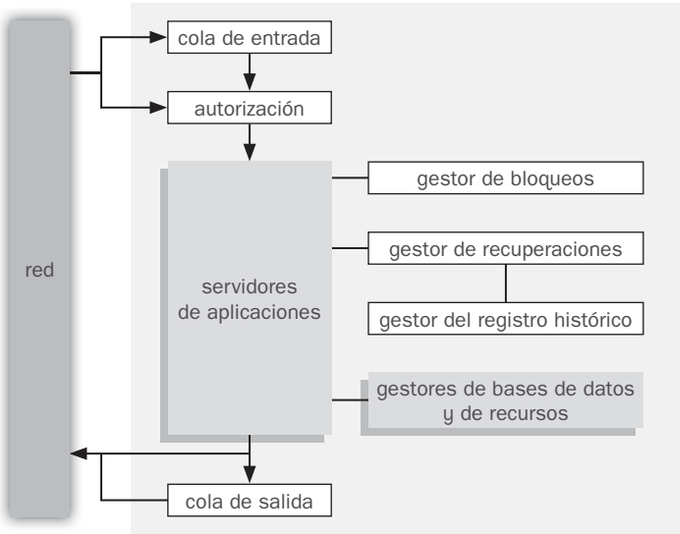


Figura 26.2. Componentes de los monitores TP.

26.1.2. Coordinación de aplicaciones mediante monitores TP

En la actualidad, las aplicaciones suelen tener que interactuar con múltiples bases de datos. Puede que tengan que interactuar con sistemas heredados, como los sistemas de almacenamiento de finalidad especial construidos directamente con base en los sistemas de archivos. Finalmente, puede que tengan que comunicarse con usuarios o con otras aplicaciones en sitios remotos. Por tanto, también tienen que interactuar con subsistemas de comunicaciones. En esos sistemas es importante poder coordinar los accesos a los datos e implementar las propiedades ACID de las transacciones.

Los monitores TP modernos proporcionan soporte para la construcción y la gestión de aplicaciones de un tamaño grande, creadas a partir de varios subsistemas como las bases de datos, los sistemas heredados y los sistemas de comunicaciones. Los monitores TP tratan cada subsistema como un **gestor de recursos** que proporciona acceso transaccional a algún conjunto de recursos. La interfaz entre el monitor TP y el gestor de recursos se define mediante un conjunto de funciones primitivas de transacción, como *begin_transaction* (iniciar transacción), *commit_transaction* (comprometer transacción), *abort_transaction* (abortar transacción) y *prepare_to_commit_transaction* (preparar para comprometer transacción, para el compromiso de dos fases). Por supuesto, el gestor de recursos también debe proporcionar otros servicios, tales como aportar datos, a la aplicación.

La interfaz del gestor de recursos se define por la norma de procesamiento de transacciones distribuidas X/Open (X/Open Distributed Transaction Processing). Muchos sistemas de bases de datos soportan los estándares X/Open y pueden actuar como gestores de recursos. Los monitores TP —amén de otros productos, como los sistemas SQL, que soportan los estándares X/Open— pueden conectarse con los gestores de recursos.

Además, los servicios proporcionados por los monitores TP, como la mensajería persistente y las colas duraderas, actúan como gestores de recursos que soportan las transacciones. Pueden actuar como coordinadores de los compromisos de dos fases para las transacciones que tienen acceso a estos servicios y a los sistemas de bases de datos. Por ejemplo, cuando se ejecuta una transacción de actualización encolada, se entrega un mensaje y se elimina la transacción solicitada de la cola de solicitudes. El compromiso de dos fases entre la base de datos y los gestores de recursos para las colas duraderas y para la mensajería persistente ayuda a asegurar que, independientemente de los fallos, se produzcan todas estas acciones o ninguna de ellas.

También se pueden usar los monitores TP para administrar sistemas complejos cliente-servidor que consten de varios servidores y gran número de clientes. El monitor TP coordina actividades como los puntos de control y los apagados del sistema. Proporciona la seguridad y la autentificación de los clientes. Administra los grupos de servidores añadiendo o eliminando servidores sin ninguna interrupción del sistema de bases de datos. Finalmente, controla el ámbito de los fallos. Si falla algún servidor, el monitor TP puede detectar ese fallo, abortar las transacciones en curso y reiniciarlas. Si falla algún nodo, el monitor TP puede migrar las transacciones a servidores de otros nodos y, una vez más, cancelar las transacciones incompletas. Cuando los nodos que fallan se reinician, el monitor TP puede gobernar la recuperación de los gestores de recursos del nodo.

Los monitores TP pueden servir para ocultar fallos de las bases de datos en los sistemas replicados; los sistemas remotos de copia de seguridad (Sección 16.9) son un ejemplo de sistemas replicados. Las solicitudes de transacciones se remiten al monitor TP, que transfiere los mensajes a una de las réplicas de la base de datos (al sitio principal, en el caso de sistemas de copia de seguridad remota). Si falla algún sitio, el monitor TP puede encaminar los mensajes de manera transparente hacia un sitio de copia de seguridad, enmascarando el fallo del primer sitio.

En los sistemas cliente-servidor, los clientes suelen interactuar con los servidores mediante un mecanismo de **llamada a procedimientos remotos** (*remote procedure call*: RPC), en el que el cliente realiza la llamada a un procedimiento, que se ejecuta realmente en el servidor, y los resultados se devuelven al cliente. En lo relativo al código cliente que invoca al RPC, la llamada tiene el mismo aspecto que la invocación a un procedimiento local. Los sistemas de monitores TP proporcionan una interfaz para **RPC transaccionales** con sus servicios. En esta interfaz el mecanismo RPC proporciona llamadas que pueden usarse para encerrar una serie de llamadas RPC dentro de una transacción. Por tanto, las actualizaciones llevadas a cabo por el RPC se ejecutan dentro del ámbito de la transacción y se pueden hacer retroceder si se produce algún fallo.

26.2. Flujos de trabajo de transacciones

Un **flujo de trabajo** es una actividad en la que varias entidades de procesamiento ejecutan diversas tareas de manera coordinada. Una **tarea** define un trabajo que hay que hacer y puede especificarse de varias maneras, incluyendo una descripción textual en un archivo o en un mensaje de correo electrónico, un formulario, un mensaje o un programa de computadora. La **entidad de procesamiento** que lleva a cabo las tareas puede ser una persona o un sistema de software (por ejemplo, un sistema de envío de correo electrónico, un programa de aplicación o un sistema gestor de bases de datos).

La Figura 26.3 muestra algunos ejemplos de flujos de trabajo. Un ejemplo sencillo es el de un sistema de correo electrónico. La entrega de un solo mensaje de correo implica varios sistemas de envío de correo que reciben y retransmiten el mensaje hasta que este alcance su destino, donde se almacena. Otros términos empleados en la literatura de bases de datos y similares para hacer referencia a los flujos de trabajo son **flujo de tareas** y **aplicaciones multisistema.** Las tareas del flujo de trabajo a veces se denominan **pasos.**

En general, los flujos de trabajo pueden implicar a una o varias personas. Por ejemplo, considere el procesamiento de un préstamo. El flujo de trabajo correspondiente aparece en la Figura 26.4. La persona que desea un préstamo rellena un formulario, que es revisado por el encargado de los préstamos. Un empleado que procesa las solicitudes de préstamos comprueba los datos del formulario, usando fuentes como las oficinas de riesgo crediticio. Cuando se ha reunido toda la información solicitada, el encargado de los préstamos puede que decida conceder el préstamo; puede que esa decisión tenga que ser aprobada por uno o más empleados de rango superior, después de lo cual se podrá conceder el préstamo. Cada persona de este flujo de trabajo realiza una tarea; en un banco que no tenga automatizada la tarea de procesamiento de los préstamos, la coordinación de las tareas suele ejecutarse pasando la solicitud del préstamo con notas y otra información adjunta de un empleado al siguiente. Otros ejemplos de flujos de trabajo son el procesamiento de notas de gastos, de pedidos de compra y de transacciones de tarjetas de crédito.

Aplicación de flujo de trabajo	Tarea habitual	Entidad típica de procesamiento
encaminamiento de correo electrónico	mensaje de correo electrónico	sistemas de envío de correo electrónico
procesamiento de préstamos	procesamiento de formularios	personas, software de aplicaciones
procesamiento de pedidos de compra	procesamiento de formularios	personas, software de aplicaciones, SGBD

Figura 26.3. Ejemplos de flujos de trabajo.

Hoy en día es más probable que toda la información relativa a un flujo de trabajo se almacene en forma digital en una o más computadoras y, con el auge de las redes, la información puede transferirse con facilidad de una computadora a otra. Por tanto, es viable que las organizaciones automaticen sus flujos de trabajo. Por ejemplo, para automatizar las tareas implicadas en el procesamiento de los préstamos, se puede almacenar la solicitud de préstamo y la información asociada en una base de datos. El propio flujo de trabajo implica, entonces, la transferencia de la responsabilidad de una persona a la siguiente y, posiblemente, incluso a programas que pueden capturar de manera automática la información necesaria. Las personas implicadas pueden coordinar sus actividades mediante el correo electrónico.

Los flujos de trabajo están llegando a ser cada vez más importantes por muchas razones dentro de las organizaciones y entre ellas. Muchas organizaciones tienen actualmente múltiples sistemas de software que necesitan trabajar juntos. Por ejemplo, cuando un empleado entra a trabajar en una organización, hay que proporcionar su información al sistema de nóminas; al sistema de bibliotecas; a los sistemas de autentificación, que permiten que el usuario inicie sesión en las computadoras; a las cuentas de un sistema que gestione las cuentas de la cafetería, etc. Las actualizaciones, como cuando el empleado cambia su estado o deja la organización, también tienen que ser propagadas a todos los sistemas.

Las organizaciones están automatizando cada vez más sus servicios; por ejemplo, un proveedor puede proporcionar un sistema automatizado para que los clientes realicen sus pedidos. Cuando se realiza un pedido, es posible que se deban llevar a cabo varias tareas, como reservar tiempo de producción para crear el producto pedido y tiempo del servicio de entrega para entregar el producto.

Para automatizar un flujo de trabajo, en general hay que abordar dos actividades. La primera es la **especificación del flujo de trabajo:** detallar las tareas que hay que ejecutar y definir los requisitos de la ejecución. El segundo problema es la **ejecución del flujo de trabajo,** que hay que llevar a cabo mientras se proporcionan las salvaguardas de los sistemas tradicionales de bases de datos relativas a corrección de los cálculos e integridad y durabilidad de los datos. Por ejemplo, no resulta aceptable que se pierda una solicitud de préstamo o una nota, ni que se procese más de una vez, debido a un fallo del sistema. La idea subyacente a los flujos de trabajo transaccionales es usar y ampliar los conceptos de las transacciones al contexto de los flujos de trabajo.

Ambas actividades se complican por el hecho de que muchas organizaciones usan varios sistemas de procesamiento de la información administrados de manera independiente que, en la mayor parte de los casos, se desarrollaron por separado para automatizar funciones diferentes. Puede que las actividades del flujo de trabajo exijan interacciones entre varios de esos sistemas, cada uno de los cuales lleva a cabo una tarea, así como interacciones con las personas.

En los últimos años se han desarrollado distintos sistemas de flujo de trabajo. Aquí se estudiarán las propiedades de los sistemas de flujo de trabajo en un nivel relativamente abstracto, sin descender a los detalles de ningún sistema concreto.

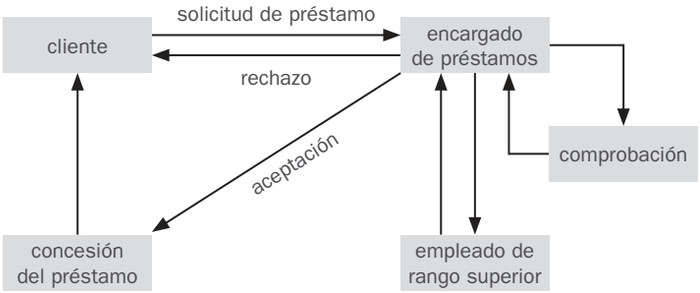


Figura 26.4. Flujo de trabajo en el procesamiento de préstamos.

26.2.1. Especificación de los flujos de trabajo

No hace falta modelar los aspectos internos de cada tarea con vistas a la especificación y gestión de un flujo de trabajo. En una vista abstracta de las tareas, cada tarea puede usar los parámetros almacenados en sus variables de entrada, recuperar y actualizar los datos del sistema local, almacenar los resultados en sus variables de salida y ser consultada sobre su estado de ejecución. En cualquier momento de la ejecución el **estado del flujo de trabajo** consiste en el conjunto de estados de las tareas constituyentes del flujo de trabajo, y los estados (valores) de todas las variables de la especificación del flujo de trabajo.

La coordinación de las tareas puede especificarse de manera estática o dinámica. La especificación estática define las tareas, y las dependencias entre ellas, antes de que comience la ejecución del flujo de trabajo. Por ejemplo, las tareas del flujo de trabajo de las notas de gastos pueden consistir en la aprobación de las notas por una secretaria, un gestor y un contable, en ese orden, y, finalmente, en la entrega de un cheque. Las dependencias entre las tareas pueden ser sencillas; hay que completar cada tarea antes de que comience la siguiente.

Una generalización de esta estrategia es la imposición de una condición previa a la ejecución de cada tarea del flujo de trabajo, de modo que todas las tareas posibles del flujo de trabajo y sus dependencias se conozcan por anticipado, pero que solo se ejecuten aquellas tareas cuyas condiciones previas se satisfagan. Las condiciones previas pueden definirse mediante dependencias como las siguientes:

- El **estado de ejecución** de otras tareas, por ejemplo, «la tarea t_i no puede comenzar hasta que la tarea t_j haya finalizado», o «la tarea t_i debe abortarse si la tarea t_j se ha comprometido».
- Los **resultados** de otras tareas, por ejemplo, «la tarea t_i puede comenzar si la tarea t_j devuelve un valor mayor que veinticinco», o «la tarea de aprobación por el gestor puede comenzar si la tarea de aprobación por la secretaria devuelve el resultado Aceptar».
- Las **variables externas** modificadas por los eventos externos, por ejemplo, «la tarea t_i no puede iniciarse antes de las nueve de la mañana», o «la tarea t_i debe iniciarse antes de que transcurran veinticuatro horas desde la finalización de la tarea t_j».

Las dependencias pueden combinarse mediante conectores lógicos **(or, and, not)** para formar condiciones previas complejas de planificación.

Un ejemplo de planificación dinámica de las tareas son los sistemas de encaminamiento del correo electrónico. La tarea que hay que programar a continuación para cada mensaje de correo depende de la dirección de destino de ese mensaje y de los encaminadores intermedios que estén en funcionamiento.

26.2.2. Requisitos de atomicidad ante fallos de los flujos de trabajo

El diseñador del flujo de trabajo puede especificar sus requisitos de **atomicidad ante fallos** de acuerdo con su semántica. El concepto tradicional de atomicidad ante fallos exige que el fallo de cualquier tarea implique el fallo del flujo de trabajo. Sin embargo, un flujo de trabajo puede, en muchos casos, sobrevivir al fallo de una de sus tareas, por ejemplo, ejecutando una tarea funcionalmente equivalente en otro sitio. Por tanto, se debe permitir al diseñador que defina los requisitos de atomicidad ante fallos del flujo de trabajo. El sistema debe garantizar que cada ejecución de un flujo de trabajo termine en un estado que satisfaga los requisitos de atomicidad ante los fallos definidos por el diseñador. Esos estados se denominan **estados de terminación aceptables** del flujo de trabajo. Todos los demás estados del flujo de trabajo constituyen un conjunto de **estados de terminación no aceptables,** en los que puede que no se cumplan los requisitos de atomicidad de los fallos.

Los estados aceptables de terminación pueden declararse comprometidos o abortados. Un **estado aceptable de terminación comprometido** es un estado de ejecución en el que se han conseguido los objetivos del flujo de trabajo. Por el contrario, un **estado aceptable de terminación abortado** es un estado válido de terminación en el que el flujo de trabajo no ha logrado alcanzar sus objetivos. Si se ha alcanzado un estado aceptable de terminación abortado, hay que deshacer todos los efectos indeseables de la ejecución parcial del flujo de trabajo de acuerdo con los requisitos de atomicidad ante fallos del flujo de trabajo.

El flujo de trabajo debe alcanzar un estado aceptable de terminación *incluso en caso de fallo del sistema*. Por tanto, si el flujo de trabajo se hallaba en un estado no aceptable de terminación en el momento del fallo, durante la recuperación del sistema hay que llevarlo a un estado aceptable de terminación (bien sea abortado, bien sea comprometido).

Por ejemplo, en el flujo de trabajo del procesamiento de los préstamos, en el estado final, o bien se comunica al solicitante del préstamo que no se le puede conceder, o se le abona el importe solicitado. En caso de fallo, como puede ser un fallo de larga duración del sistema de verificación, puede devolverse la solicitud de préstamo al solicitante con una explicación adecuada; este resultado constituiría una terminación abortada aceptable. Una terminación comprometida aceptable sería la aceptación o el rechazo de la solicitud.

En general, las tareas pueden comprometer y liberar sus recursos antes de que el flujo de trabajo alcance un estado de terminación. Sin embargo, si la transacción multitarea aborta posteriormente, su atomicidad ante fallos puede que exija que se deshagan todos los efectos de las tareas ya completadas (por ejemplo, las subtransacciones comprometidas) ejecutando tareas compensadoras (como las subtransacciones). La semántica de la compensación exige que la transacción compensadora acabe completando su ejecución con éxito, quizás tras varios reenvíos.

En el flujo de trabajo del procesamiento de las notas de gastos, por ejemplo, puede que se reduzca el importe del presupuesto del departamento debido a la aprobación inicial de una nota de gastos por el gestor. Si posteriormente se rechaza esa nota, a causa de un fallo o por otro motivo, es posible que haya que restaurar el presupuesto mediante una transacción compensadora.

26.2.3. Ejecución de los flujos de trabajo

La ejecución de las tareas puede ser controlada por un coordinador humano o por un sistema de software denominado **sistema gestor de flujos de trabajo.** Los sistemas gestores de flujos de trabajo constan de un planificador, los agentes para las tareas y un mecanismo para consultar el estado del sistema del flujo de trabajo. Cada agente de tarea controla la ejecución de una tarea por una entidad de procesamiento. El planificador es un programa que procesa los flujos de trabajo remitiendo diferentes tareas para su ejecución, controlando los diferentes eventos y evaluando las condiciones relativas a las dependencias entre las tareas. El planificador puede remitir una tarea para su ejecución (a un agente de tareas) o solicitar que se aborte una tarea previamente remitida. En el caso de las transacciones con múltiples bases de datos, las tareas son subtransacciones y las entidades de procesamiento son sistemas gestores de bases de datos locales. De acuerdo con las especificaciones del flujo de trabajo, el planificador hace que se cumplan las dependencias de planificación y es responsable de asegurar que las tareas alcancen estados aceptables de terminación.

En la arquitectura de los sistemas gestores de flujos de trabajo hay tres alternativas de desarrollo. La **arquitectura centralizada** tiene un solo planificador que programa las tareas de todos los flujos de trabajo que se ejecutan de manera concurrente. La **arquitectura parcialmente distribuida** tiene un planificador para cada flujo de trabajo. Cuando los problemas de la ejecución concurrente pueden diferenciarse de la función de planificación, esta opción es una elección natural. La **arquitectura completamente distribuida** no tiene planificador, pero los agentes de tareas coordinan su ejecución comunicándose entre sí para satisfacer las dependencias entre las tareas y otros requisitos de ejecución del flujo de trabajo.

Los sistemas de ejecución de flujos de trabajo más sencillos siguen el enfoque totalmente distribuido que se acaba de describir y se basan en la mensajería. La mensajería puede implementarse mediante mecanismos de mensajería persistente para garantizar la entrega. Algunas implementaciones usan el correo electrónico para la mensajería; estas implementaciones proporcionan muchas de las características de la mensajería persistente, pero generalmente no garantizan la atomicidad de la entrega de los mensajes ni el compromiso de las transacciones. Cada sitio tiene un agente de tareas que ejecuta las tareas recibidas mediante los mensajes. Puede que la ejecución también implique la entrega de mensajes a personas que tienen que llevar a cabo alguna acción. Cuando se completa una tarea en un sitio y hay que procesarla en otro sitio, el agente de tareas transmite un mensaje al sitio siguiente. El mensaje contiene toda la información relevante sobre la tarea que hay que realizar. Estos sistemas de flujos de trabajo basados en mensajes resultan especialmente útiles en las redes que se pueden desconectar durante parte del tiempo.

El enfoque centralizado se usa en sistemas de flujos de trabajo en los que los datos se almacenan en una base de datos central. El planificador notifica a los diferentes agentes, como pueden ser las personas o los programas informáticos, que hay que llevar a cabo una tarea y realiza un seguimiento de su finalización. Resulta más sencillo realizar un seguimiento del estado del flujo de trabajo con el enfoque centralizado que con el enfoque completamente distribuido.

El planificador debe garantizar que termine el flujo de trabajo en uno de los estados aceptables de terminación especificados. Idealmente, antes de intentar ejecutar un flujo de trabajo el planificador debe examinarlo para comprobar si puede terminar en un estado no aceptable. Si el planificador no puede garantizar que el flujo de trabajo termine en un estado aceptable, debe rechazar esas especificaciones sin intentar ejecutar el flujo de trabajo. Por ejemplo, considere un flujo de trabajo consistente en dos tareas representadas por las subtransacciones S_1 y S_2, con los requisitos de atomicidad ante fallos que indican que se deben comprometer las dos subtransacciones o ninguna de ellas. Si S_1 y S_2 no proporcionan estados preparados para comprometerse (para un compromiso de dos fases) y, además, no tienen transacciones compensadoras, es posible alcanzar un estado en el que se comprometa una subtransacción y se aborte la otra, y no haya manera de llevar a las dos al mismo estado. Por tanto, esa especificación del flujo de trabajo es **insegura** y debe rechazarse.

Los controles de seguridad, como el que se acaba de describir, pueden ser imposibles o poco prácticos de implementar en el planificador; pasa a ser, entonces, responsabilidad de la persona que diseña la especificación del flujo de trabajo asegurarse de que este sea seguro.

26.2.4. Recuperación en los flujos de trabajo

El objetivo de la **recuperación en los flujos de trabajo** es hacer que se cumpla la atomicidad ante fallos. Los procedimientos de recuperación deben asegurarse de que si se produce un fallo en cualquiera de los componentes de procesamiento del flujo de trabajo (incluido el planificador), este acabe alcanzando un estado aceptable de terminación (sea abortado o comprometido). Por ejemplo, el planificador puede continuar procesando tras el fallo y

la recuperación, como si no hubiera pasado nada, lo que proporciona recuperabilidad hacia delante. En caso contrario, el planificador puede abortar todo el flujo de trabajo (es decir, alcanzar uno de los estados globales abortados). En cualquiera de los casos, puede que haga falta comprometer algunas subtransacciones o incluso remitirlas para su ejecución (por ejemplo, las subtransacciones compensadoras).

Se supone que las entidades de procesamiento implicadas en el flujo de trabajo tienen sus propios sistemas locales de recuperación y tratan sus fallos locales. Para recuperar el contexto del entorno de ejecución, las rutinas de recuperación de los fallos deben restaurar la información de estado del planificador en el momento del fallo, incluida la información sobre el estado de ejecución de cada tarea. Por tanto, la información de estado correspondiente debe registrarse en almacenamiento estable.

También hay que considerar el contenido de las colas de mensajes. Cuando un agente transfiere una tarea a otro, la transferencia debe ejecutarse exactamente una vez: si la transferencia tiene lugar dos veces, puede que se ejecute dos veces una tarea; si no se produce la transferencia, es posible que se pierda la tarea. La mensajería persistente (Sección 19.4.3) proporciona exactamente las características para asegurar una transferencia positiva y única.

26.2.5. Sistemas gestores de flujos de trabajo

Los flujos de trabajo suelen codificarse a mano como parte de los sistemas de aplicaciones. Por ejemplo, los sistemas de planificación de los recursos de las empresas (*enterprise resource planning*: ERP), que ayudan a coordinar las actividades en toda la empresa, tienen incorporados numerosos flujos de trabajo.

El objetivo de los sistemas gestores de flujos de trabajo es simplificar la construcción de flujos de trabajo y hacerlos más dignos de confianza, permitiendo que se especifiquen en un alto nivel y se ejecuten de acuerdo con la especificación. Hay gran número de sistemas comerciales de gestión de flujos de datos; algunos, como FlowMark de IBM, son sistemas gestores de flujos de trabajo de propósito general, mientras que otros son específicos de flujos de trabajo concretos, como los sistemas de procesamiento de comandos o los sistemas de comunicación de fallos.

En el mundo actual de organizaciones interconectadas, no es suficiente gestionar los flujos de trabajo exclusivamente en el interior de una organización. Los flujos de trabajo que atraviesan las fronteras organizativas se están volviendo cada vez más frecuentes. Por ejemplo, considere un pedido realizado por una organización y comunicado a otra organización que lo atiende. En cada organización puede que haya un flujo de trabajo asociado con el pedido y es importante que los flujos de trabajo puedan operar entre sí con objeto de minimizar la intervención humana.

Se utiliza el término **gestión de procesos de negocio** para referirse a la gestión de flujos de trabajo relacionada con los procesos de negocio. En la actualidad, las aplicaciones van exponiendo cada vez más sus funciones como servicios que se pueden invocar desde otras aplicaciones, a menudo usando una arquitectura de servicios web. Una arquitectura de sistemas basada en la invocación de servicios que proporcionan múltiples aplicaciones se denomina **arquitectura orientada a servicios, SOA.** Estos servicios son el nivel base sobre el que en la actualidad se implementan los flujos de trabajo. La lógica del proceso que controla el flujo de trabajo invocando los procesos se denomina **orquestación.**

Entre los sistemas de gestión de procesos de negocio basados en la arquitectura SOA están BizTalk Server de Microsoft, WebSphere Business Integration Server Foundation, de IBM, y WebLogic Process Edition, de BEA, entre otros.

El **lenguaje de ejecución de procesos de negocio en servicios web (WS-BPEL)** es una norma basada en XML para la especificación de servicios web y procesos de negocio (flujos de trabajo) basados en servicios web, que se pueden ejecutar mediante un sistema de gestión de procesos de negocio. La **notación de modelado de procesos de negocio BPMN** (Business Process Modeling Notation), es una norma para el modelado gráfico de los procesos de negocio en un flujo de trabajo, y el **lenguaje de definición de procesos XML** es una representación en XML de las definiciones de procesos de negocio, basada en los diagramas BPMN.

26.3. Comercio electrónico

El término comercio electrónico (*e-commerce*) hace referencia al proceso de llevar a cabo varias actividades relacionadas con el comercio mediante medios electrónicos, principalmente por Internet. Entre los tipos de actividades figuran:

- Actividades previas a la venta, necesarias para informar al posible comprador del producto o servicio que se desea vender.
- El proceso de venta, que incluye las negociaciones sobre el precio y la calidad del servicio, así como otros asuntos contractuales.
- El mercado: cuando hay varios vendedores y varios compradores para un mismo producto, un mercado, como la bolsa, ayuda a negociar el precio que se va a pagar por el producto. Las subastas se usan cuando hay un único vendedor y varios compradores, y las subastas inversas se usan cuando hay un solo comprador y varios vendedores.
- El pago por la compra.
- Las actividades relacionadas con la entrega del producto o servicio. Algunos productos y servicios pueden entregarse por Internet; para otros, Internet solo se usa para facilitar información sobre el envío y realizar un seguimiento de los envíos de los productos.
- Atención al cliente y servicio posventa.

Las bases de datos se usan ampliamente para soportar estas actividades. Para algunas de ellas, el uso de las bases de datos es directo, pero hay aspectos interesantes de desarrollo de aplicaciones para las demás actividades.

26.3.1. Catálogos electrónicos

Cualquier sitio de comercio electrónico proporciona a los usuarios un catálogo de los productos y servicios que ofrece. Los servicios facilitados por los catálogos electrónicos pueden variar considerablemente.

Como mínimo, un catálogo electrónico debe proporcionar servicios de exploración y de búsqueda para ayudar a los clientes a encontrar el producto que buscan. Para ayudar en la exploración conviene que los productos estén organizados en una jerarquía intuitiva, de modo que unas pocas pulsaciones en los hipervínculos puedan llevar a los clientes a los productos en los que estén interesados. Las palabras clave facilitadas por el cliente (por ejemplo, «cámara digital» o «computadora») deben acelerar el proceso de búsqueda de los productos solicitados. Los catálogos electrónicos también deben proporcionar un medio para que los clientes comparen con facilidad las alternativas a la hora de elegir entre los diferentes productos.

Los catálogos electrónicos se pueden personalizar para los clientes. Por ejemplo, puede que un vendedor al por menor tenga un acuerdo con una gran empresa para venderle algunos productos con descuento. Un empleado de la empresa, al examinar el catálogo para adquirir productos para la empresa, debería ver los precios con el descuento acordado, en vez de con los precios normales. Debido a las restricciones legales sobre las ventas de algunos tipos de

artículos, no se deberían mostrar a los clientes menores de edad, o de ciertos Estados o países, los artículos que no se les pueden vender legalmente. Los catálogos también se pueden personalizar para usuarios individuales, de acuerdo con su historial de compras. Por ejemplo, se pueden ofrecer descuentos especiales a los clientes frecuentes en algunos productos.

El soporte de esa personalización necesita que la información de los clientes y la información sobre precios o descuentos especiales y sobre restricciones a las ventas se guarden en una base de datos. También hay desafíos en el soporte de tasas de transacciones muy elevadas, que suelen abordarse guardando en la caché los resultados de las consultas o las páginas web generadas.

26.3.2. Mercados

Cuando hay varios vendedores o varios compradores (o ambas cosas) para un producto, los mercados ayudan a negociar el precio que debe pagarse por él. Hay varios tipos diferentes de mercados:

- En los sistemas de **subasta inversa,** los compradores manifiestan sus necesidades y los vendedores pujan por proporcionar el artículo. El proveedor que ofrece el precio más bajo gana la subasta. En los sistemas de puja cerrada las pujas no se hacen públicas, mientras que en los sistemas de puja abierta las pujas sí se hacen públicas.
- En las **subastas** hay varios compradores y un solo vendedor. Por simplificar, suponga que solo se vende un ejemplar de cada artículo. Los compradores pujan por los artículos que se venden y el que puja más alto consigue comprarlo por el precio de la puja.

  Cuando hay varios ejemplares de un artículo las cosas se vuelven más complicadas: suponga que hay cuatro artículos y puede que un comprador desee tres ejemplares a 10 € cada uno, mientras que otro desee dos copias por 13 € cada una. No es posible satisfacer ambas pujas. Los artículos carecen de valor si no se venden (por ejemplo, billetes de avión, que deben venderse antes de que despegue el avión), por lo que el vendedor simplemente selecciona un conjunto de pujas que maximice los ingresos. En caso contrario, la decisión es más complicada.
- En las **bolsas,** como puede ser una bolsa de valores, hay varios vendedores y varios compradores. Los compradores pueden especificar el precio máximo que desean pagar, mientras que los vendedores especifican el precio mínimo que desean obtener. Suele haber un *creador de mercado* que casa las ofertas de compra y de venta y decide el precio de cada intercambio (por ejemplo, al precio de la oferta de venta).

  Existen otros tipos de mercados más complejos.

Entre los aspectos de las bases de datos relacionados con el manejo de los mercados figuran:

- Hay que autentificar a los que realizan las pujas antes de permitirles pujar.
- Las pujas (de compra o de venta) deben registrarse de modo seguro en una base de datos. Hay que comunicar rápidamente las pujas al resto de las personas implicadas en el mercado (como pueden ser todos los compradores o todos los vendedores), que puede que sean numerosas.
- Los retrasos en la difusión de las pujas pueden llevar a pérdidas financieras para algunos participantes.
- Los volúmenes de los intercambios pueden resultar tremendamente grandes en tiempos de volatilidad de las bolsas, o hacia el final de las subastas. Por tanto, para estos sistemas se utilizan bases de datos de muy alto rendimiento con elevados grados de paralelismo.

26.3.3. Liquidación de pedidos

Una vez seleccionados los artículos (quizás mediante un catálogo electrónico) y determinado el precio (quizás mediante un mercado electrónico), hay que liquidar el pedido. La liquidación implica el pago y la entrega de las mercancías.

Una manera sencilla pero poco segura de pagar electrónicamente consiste en enviar el número de una tarjeta de crédito. Hay dos problemas principales. En primer lugar, es posible el fraude con las tarjetas de crédito. Cuando un comprador paga mercancías físicas las empresas pueden asegurarse de que la dirección de entrega coincide con la del titular de la tarjeta, de modo que nadie más reciba la mercancía, pero para las mercancías entregadas de manera electrónica no es posible realizar esa comprobación. En segundo lugar, hay que confiar en que el vendedor solo facture el artículo acordado y en que no pase el número de la tarjeta de crédito a personas no autorizadas que lo puedan usar de manera no adecuada.

Se dispone de varios protocolos para los pagos seguros que evitan los dos problemas mencionados. Además, proporcionan una mayor intimidad, ya que no hay que dar al vendedor datos innecesarios del comprador y no se le facilitan a la compañía de la tarjeta de crédito información innecesaria sobre los artículos comprados. Toda la información transmitida debe cifrarse de modo que nadie que intercepte los datos por la red pueda averiguar su contenido. El cifrado con claves pública y privada se usa mucho para esta tarea.

Los protocolos también deben evitar los **ataques de hombre en medio,** en los que alguien puede suplantar al banco o a la compañía de la tarjeta de crédito, o incluso al vendedor o al comprador, y sustraer información secreta. La suplantación puede realizarse pasando una clave falsa como si fuera la clave pública de otra persona (el banco, la compañía de la tarjeta de crédito, el comerciante o el comprador). La suplantación se evita mediante un sistema de **certificados digitales,** en los que las claves públicas vienen firmadas por una agencia de certificación cuya clave pública es bien conocida (o que, a su vez, tiene su clave pública certificada por otra agencia de certificación, y así hasta llegar a una clave que sea bien conocida). A partir de la clave pública bien conocida el sistema puede autentificar las otras claves comprobando los certificados en una secuencia inversa. Los certificados digitales se describieron anteriormente en la Sección 9.8.3.2.

En el comienzo de la web se desarrollaron varios sistemas de pago. Uno de ellos fue un protocolo de pago seguro denominado *transacciones electrónicas seguras* (*secure electronic transaction*: SET). El protocolo necesita varias rondas de comunicación entre el comprador, el vendedor y el banco para garantizar la seguridad de la transacción. También hay sistemas que ofrecen más anonimato, parecido al proporcionado por el dinero físico en efectivo. El sistema de pagos *DigiCash* es uno de esos sistemas. Cuando se realiza un pago en estos sistemas no es posible identificar al comprador. Sin embargo, la identificación del comprador resulta muy sencilla con las tarjetas de crédito e, incluso en el caso de SET, es posible identificar al comprador con la colaboración de la compañía de la tarjeta de crédito o del banco. Sin embargo, ninguno de estos sistemas tuvo éxito comercialmente, tanto por razones técnicas como de otro tipo.

En la actualidad muchos bancos proporcionan **pasarelas de pago seguro** que permiten a un comprador pagar en línea en el sitio web de un banco, sin exponer la tarjeta de crédito ni la información de la cuenta bancaria al vendedor. Cuando se realiza una compra en línea, el navegador web del comprador se redirige a la pasarela para completar el pago proporcionando la información de la tarjeta de crédito o de la cuenta bancaria, tras lo cual se vuelve a redirigir al comprador al sitio del vendedor para completar la compra. Al contrario que los protocolos SET o DigiCash, no existe ningún software ejecutándose en la máquina del comprador, excepto el navegador web; como resultado este enfoque ha encontrado mucha mayor aceptación y los anteriores han sido desechados.

Un enfoque alternativo es el que utiliza el sistema PayPal, en el que tanto el comprador como el vendedor tienen cuenta en una plataforma común, y la transferencia del dinero se produce en su totalidad dentro de la plataforma común. El comprador carga en su cuenta el dinero usando una tarjeta de crédito y puede transferir el dinero a la cuenta del vendedor. Este enfoque ha tenido mucho éxito con vendedores pequeños, ya que no requiere que ni el comprador ni el vendedor ejecuten ningún software.

26.4. Bases de datos en memoria principal

Para permitir una velocidad elevada de procesamiento de transacciones (centenares o millares de transacciones por segundo) hay que usar un hardware de alto rendimiento y aprovechar el paralelismo; no obstante, estas técnicas, por sí solas, resultan insuficientes para obtener tiempos de respuesta muy bajos, ya que las operaciones de E/S de disco siguen constituyendo un cuello de botella; se necesitan alrededor de diez milisegundos para cada operación de E/S, y esta cifra no se ha reducido a una velocidad comparable con el aumento de la velocidad de los procesadores. Las operaciones de E/S suelen ser el cuello de botella de las operaciones de lectura y de los compromisos de las transacciones. La elevada latencia de los discos (alrededor de diez milisegundos de promedio) no solo aumenta el tiempo necesario para tener acceso a un elemento de datos, sino que también limita el número de accesos por segundo.[1]

Se puede hacer un sistema de bases de datos menos ligado a los discos aumentando el tamaño de la memoria intermedia de la base de datos. Los avances en la tecnología de la memoria principal permiten construir memorias principales de gran tamaño con un coste relativamente bajo. Hoy en día los sistemas comerciales de 64 bits admiten memorias principales de decenas de gigabytes. Una base de datos en memoria principal es Oracle TimesTen. Consulte las referencias bibliográficas para más información sobre bases de datos en memoria principal.

Para algunas aplicaciones, como el control en tiempo real, es necesario almacenar los datos en la memoria principal para cumplir los requisitos de rendimiento. El tamaño de memoria exigido para la mayoría de estos sistemas no resulta excepcionalmente grande, aunque hay unas cuantas aplicaciones que exigen que residan en la memoria varios gigabytes de datos. Dado que el tamaño de la memoria ha estado creciendo a una velocidad muy elevada, se puede suponer que cada vez los datos de más aplicaciones puedan caber en la memoria principal.

Las memorias principales de gran tamaño permiten el procesamiento más rápido de las transacciones, ya que los datos residen en la memoria. No obstante, sigue habiendo limitaciones relacionadas con los discos:

- Hay que guardar en almacenamiento estable los registros del registro histórico antes de comprometer una transacción. El rendimiento mejorado que hace posible la memoria principal de gran tamaño puede hacer que el proceso de registro se convierta en un cuello de botella. Se puede reducir el tiempo de compromiso creando un búfer de registro estable en la memoria principal, usando RAM no volátil (implementada, por ejemplo, mediante memoria sustentada por baterías). La sobrecarga impuesta por el registro también puede reducirse mediante la técnica de *compromiso en grupo* estudiada más adelante en este apartado. El rendimiento (el número de transacciones por segundo) sigue estando limitado por la velocidad de transferencia de datos del disco de registro.
- Sigue habiendo que escribir los bloques de la memoria intermedia marcados como modificados por las transacciones comprometidas para que se reduzca la cantidad de registro histórico que hay que volver a ejecutar en el momento de la recuperación. Si la velocidad de actualización es extremadamente elevada, la velocidad de transferencia de los datos al disco puede convertirse en un cuello de botella.
- Si el sistema falla, se pierde toda la memoria principal. En la recuperación el sistema tiene la memoria intermedia de la base de datos vacía y hay que introducir desde el disco los elementos de datos cuando se tenga acceso a ellos. Por tanto, incluso una vez que esté completa la recuperación hace falta algo de tiempo antes de que se cargue completamente la base de datos en la memoria principal y se pueda reanudar el procesamiento de transacciones de alta velocidad.

Por otro lado, las bases de datos en memoria principal proporcionan oportunidades para la optimización:

- Como la memoria resulta más costosa que el espacio de disco, hay que diseñar las estructuras internas de los datos de la memoria principal para reducir los requisitos de espacio. No obstante, las estructuras de datos pueden tener punteros que atraviesen varias páginas, a diferencia de los de las bases de datos en disco, en las que el coste de que la operación de E/S atraviese varias páginas resultaría excesivamente elevado. Por ejemplo, las estructuras arbóreas de las bases de datos en memoria principal pueden ser relativamente profundas, a diferencia de los árboles B^+, pero deben minimizar los requisitos de espacio.

  Sin embargo, la diferencia de velocidad entre la memoria caché y la memoria principal, y el hecho de que los datos se transfieren entre ambas en unidades de *líneas de caché* (normalmente unos 64 bytes), genera una situación en la que la relación entre la memoria caché y la memoria principal no es distinta de la relación entre la memoria principal y el disco (aunque con menores diferencias de velocidad). En este sentido se han encontrado muy útiles los árboles B^+ con nodos pequeños que quepan en una línea de caché incluso en bases de datos en memoria principal.
- No hace falta fijar en la memoria las páginas de la memoria intermedia antes de que se tenga acceso a los datos, ya que las páginas de la memoria intermedia no se sustituyen nunca.
- Las técnicas de procesamiento de consultas deben diseñarse para minimizar la sobrecarga de espacio, de modo que no se superen los límites de la memoria mientras se evalúa una consulta; esa situación daría lugar a que se paginara el área de intercambio y se ralentizara el procesamiento de la consulta.
- Una vez eliminado el cuello de botella de las operaciones de E/S del disco, operaciones como los bloqueos y los pestillos pueden convertirse en cuellos de botella. Hay que eliminar estos cuellos de botella mediante mejoras en la implementación de estas operaciones.
- Los algoritmos de recuperación pueden optimizarse, ya que rara vez hace falta borrar las páginas para hacer sitio a otras páginas.

El proceso de comprometer una transacción *T* exige que se escriban en almacenamiento estable estos registros:

- Todos los registros del registro histórico asociados con *T* que no se hayan remitido al almacenamiento estable.
- El registro <**comprometer** T> del registro histórico.

Estas operaciones de salida suelen exigir la salida de bloques que solo se hallan parcialmente llenos. Para asegurarse de que se saquen bloques casi llenos se usa la técnica de **compromiso en grupo.** En lugar de intentar comprometer *T* cuando se complete *T*, el sistema espera hasta que se hayan completado varias transacciones, o hasta que haya pasado un determinado periodo de tiempo desde que se completó la ejecución de la transacción. Luego compromete el grupo de transacciones que están esperando, todas juntas. Los bloques es-

1 La latencia de escritura para memorias flash depende de si se debe realizar primero o no una operación de borrado.

critos en el registro histórico en almacenamiento estable contienen registros de varias transacciones. Mediante una cuidadosa selección del tamaño del grupo y del tiempo máximo de espera, el sistema puede asegurarse de que los bloques estén llenos cuando se escriben en el almacenamiento estable sin hacer que las transacciones esperen demasiado. Esta técnica da como resultado, en promedio, menos operaciones de salida por cada transacción comprometida.

Aunque el compromiso en grupo reduce la sobrecarga impuesta por el registro histórico, da lugar a un ligero retraso en el compromiso de las transacciones que llevan a cabo actualizaciones. El retraso puede hacerse bastante pequeño (del orden de diez milisegundos), lo que resulta aceptable para muchas aplicaciones. Estos retrasos pueden eliminarse si los discos o los controladores de disco soportan los búferes de RAM no volátil para las operaciones de escritura. Las transacciones pueden comprometerse en cuanto la operación de escritura se lleve a cabo en la memoria intermedia de RAM no volátil. En este caso, no hay necesidad de compromiso en grupo.

Obsérvese que el compromiso en grupo resulta útil incluso en bases de datos con datos residentes en disco, no solo para las bases de datos en memoria principal. Si se utiliza memoria flash en lugar de discos magnéticos para el almacenamiento de los registros históricos, el retardo de compromiso se reduce significativamente. Sin embargo, el compromiso en grupo puede seguir siendo útil, ya que minimiza el número de páginas que se escriben; esto se traslada a ventajas en el rendimiento de las memorias flash, ya que las páginas no se pueden sobrescribir y las operaciones de borrado son costosas. (Los sistemas de memoria flash reasignan las páginas lógicas a una página física previamente borrada, evitando el retardo en el momento en que una página se escribe, pero las operaciones de borrado eventualmente deben ser realizadas como parte del proceso de recolección de antiguas páginas).

26.5. Sistemas de transacciones de tiempo real

Las restricciones de integridad que se han considerado hasta ahora corresponden a los valores almacenados en la base de datos. En determinadas aplicaciones, las restricciones incluyen **plazos** en los que se tiene que haber completado una tarea. Entre estas aplicaciones están la gestión de factorías, el control del tráfico y la planificación. Cuando se incluyen plazos, la corrección de la ejecución ya no es exclusivamente un problema de consistencia de la base de datos. Por el contrario, hay que preocuparse por el número de plazos sobrepasados y por el tiempo que hace que se sobrepasaron. Los plazos se clasifican de la siguiente manera:

- **Plazo estricto.** Pueden producirse problemas graves, como fallos del sistema, si no se completa una tarea antes de su plazo.
- **Plazo firme.** La tarea no tiene ningún valor si se completa después del plazo.
- **Plazo flexible.** La tarea tiene un valor decreciente si se completa tras el plazo, y el valor se aproxima a cero a medida que aumenta el retraso.

Los sistemas con plazos se denominan **sistemas de tiempo real.**

La gestión de transacciones en los sistemas de tiempo real debe tener en cuenta los plazos. Si el protocolo de control de concurrencia determina que la transacción T_i debe esperar, puede hacer que T_i supere el plazo. En esos casos, puede que resulte preferible adelantar la transacción que mantiene el bloqueo y permitir que T_i siga adelante. El adelantamiento debe usarse con cuidado, no obstante, ya que el tiempo perdido por la transacción adelantada (debido al retroceso y al reinicio) puede hacer que la transacción supere su plazo. Por desgracia, es difícil determinar si es preferible retroceder o esperar en una situación dada.

Una de las principales dificultades para disponer de restricciones de tiempo real surge de la variabilidad en el tiempo de ejecución de las transacciones. En el caso más favorable, todos los accesos a los datos hacen referencia a datos de la memoria intermedia de la base de datos. En el peor de los casos, cada acceso hace que se escriba una página de la memoria intermedia en el disco (precedida de los registros del registro histórico necesario), seguido de la lectura desde el disco de la página que contiene los datos a los que hay que tener acceso. Como los dos o más accesos al disco requeridos en el peor de los casos tardan varios órdenes de magnitud más que las referencias a la memoria principal necesarias en el caso más favorable, el tiempo de ejecución de las transacciones puede estimarse con muy poca precisión si los datos residen en el disco. Por tanto, si hay que cumplir restricciones en tiempo real se suelen usar las bases de datos de memoria principal.

Sin embargo, aunque los datos residan en la memoria principal, la variabilidad en el tiempo de ejecución surge de las esperas de los bloqueos, de los abortos de las transacciones, etc. Los investigadores han dedicado esfuerzos considerables al control de concurrencia para las bases de datos de tiempo real. Han ampliado los protocolos de bloqueo para conceder una prioridad más elevada a las transacciones con plazos más próximos y han determinado que los protocolos de concurrencia optimistas tienen un buen comportamiento en las bases de datos de tiempo real; es decir, estos protocolos incluso dan lugar a menos plazos sobrepasados que los protocolos de bloqueo ampliados. En las notas bibliográficas se proporcionan referencias sobre las investigaciones en el área de las bases de datos de tiempo real.

En los sistemas de tiempo real, los plazos, y no la velocidad absoluta, son el aspecto más importante. El diseño de sistemas de tiempo real implica asegurarse de que hay suficiente capacidad de procesamiento como para respetar los plazos sin necesitar excesivos recursos de hardware. La consecución de este objetivo, pese a la variabilidad de los tiempos de ejecución resultante de la gestión de las transacciones, sigue constituyendo un problema sin resolver.

26.6. Transacciones de larga duración

El concepto de transacción se desarrolló inicialmente en el contexto de las aplicaciones de procesamiento de datos, en el que la mayor parte de las transacciones son de corta duración y no interactivas. Aunque las técnicas presentadas aquí y, anteriormente, en los Capítulos 14, 15 y 16 funcionan bien en esas aplicaciones, surgen problemas graves cuando se aplica este concepto a sistemas de bases de datos que implican la interacción con personas. Esas transacciones tienen las siguientes propiedades principales:

- **Larga duración.** Una vez que una persona interactúa con una transacción activa esa transacción se transforma en una **transacción de larga duración** desde la perspectiva de la computadora, ya que el tiempo de respuesta de las personas es lento en comparación con la velocidad de las computadoras. Además, en las aplicaciones de diseño, la actividad humana puede suponer horas, días o periodos incluso más prolongados. Por tanto, las transacciones pueden ser de larga duración en términos humanos, además de serlo en términos de la máquina.
- **Exposición de datos no comprometidos.** Los datos generados y mostrados a los usuarios por las transacciones de larga duración no están comprometidos, ya que la transacción puede abortarse. Por tanto, los usuarios, y en consecuencia, las demás transacciones, pueden verse forzados a leer datos no comprometidos. Si varios usuarios están colaborando en un proyecto puede que las transacciones de los usuarios necesiten intercambiar datos antes de comprometer las transacciones.

- **Subtareas.** Cada transacción interactiva puede consistir en un conjunto de subtareas iniciadas por el usuario. Puede que el usuario desee abortar una subtarea sin hacer necesariamente que aborte toda la transacción.
- **Recuperabilidad.** Resulta inaceptable abortar una transacción interactiva de larga duración debido a un fallo del sistema. La transacción activa debe recuperarse hasta el estado en que estuviera poco antes del fallo para que se pierda una cantidad de trabajo humano relativamente pequeña.
- **Rendimiento.** El buen rendimiento de los sistemas interactivos de transacciones se define como tiempo de respuesta rápido. Esta definición difiere de la de los sistemas no interactivos, en los que el objetivo es una productividad (número de transacciones por segundo) elevada. Los sistemas con productividad elevada hacen un uso eficiente de los recursos del sistema. Sin embargo, en el caso de las transacciones interactivas, el recurso más costoso es el usuario. Si hay que optimizar la eficiencia y la satisfacción del usuario, el tiempo de respuesta debe ser rápido (desde el punto de vista humano). En los casos en los que una tarea tarde mucho tiempo, el tiempo de respuesta debe ser predecible (es decir, la variabilidad de los tiempos de respuesta debe ser baja), de modo que los usuarios puedan administrar bien su tiempo.

En las Secciones 26.6.1 a 26.6.5 se verá el motivo por el que estas cinco propiedades son incompatibles con las técnicas presentadas hasta ahora, y se estudiará el modo en que se pueden modificar dichas técnicas para acomodar las transacciones interactivas de larga duración.

26.6.1. Ejecuciones no secuenciables

Las propiedades que se han estudiado hacen poco práctico obligar a que se cumpla el requisito empleado en los capítulos anteriores de que solo se permitan las planificaciones secuenciables. Cada uno de los protocolos de control de concurrencia del Capítulo 15 tiene efectos negativos sobre las transacciones de larga duración:

- **Bloqueo de dos fases.** Cuando no se puede conceder un bloqueo, la transacción que lo ha solicitado se ve obligada a esperar a que se desbloquee el elemento de datos en cuestión. La duración de la espera es proporcional a la duración de la transacción que sostiene el bloqueo. Si el elemento de datos está bloqueado por una transacción de corta duración, se espera que el tiempo de espera sea breve (excepto en el caso de interbloqueos o de carga extraordinaria del sistema). Sin embargo, si el elemento de datos está bloqueado por una transacción de larga duración, la espera será prolongada. Los tiempos de espera elevados provocan tiempos de respuesta superiores y una mayor posibilidad de interbloqueos.
- **Protocolos basados en grafos.** Los protocolos basados en grafos permiten que se liberen los bloqueos antes que con los protocolos de bloqueo de dos fases, y evitan los interbloqueos. Sin embargo, imponen una ordenación de los elementos de datos. Las transacciones deben bloquear los elementos de datos de manera consistente con esta ordenación. En consecuencia, puede que una transacción tenga que bloquear más datos de los que necesita. Además, la transacción debe mantener el bloqueo hasta que no haya posibilidad de que se vuelva a necesitar. Por tanto, es probable que se produzcan esperas por bloqueos de larga duración.
- **Protocolos basados en marcas temporales.** Los protocolos de marcas temporales nunca necesitan que las transacciones esperen. Sin embargo, exigen que las transacciones se aborten bajo ciertas circunstancias. Si se aborta una transacción de larga duración, se pierde una cantidad de trabajo sustancial. Para las transacciones no interactivas este trabajo perdido supone un problema de rendimiento. Para las transacciones interactivas el problema también es de satisfacción de los usuarios. Resulta muy poco deseable que el usuario descubra que se han deshecho varias horas de trabajo.
- **Protocolos de validación.** Al igual que los protocolos basados en las marcas temporales, los protocolos de validación hacen que se cumpla la secuencialidad abortando transacciones.

Por tanto, parece que el cumplimiento de la secuencialidad provoca esperas de larga duración, el aborto de transacciones de larga duración, o ambas cosas. Hay resultados teóricos, citados en las notas bibliográficas, que sustentan esta conclusión.

Cuando se consideran los problemas de las recuperaciones surgen dificultades adicionales con el cumplimiento de la secuencialidad. Ya se ha estudiado el problema de los retrocesos en cascada, en los que abortar una transacción puede conducir a abortar otras transacciones. Este fenómeno no es deseable, especialmente para las transacciones de larga duración. Si se usa el bloqueo, se deben mantener bloqueos exclusivos hasta el final de la transacción, si hay que evitar el retroceso en cascada. Este mantenimiento de bloqueos exclusivos, no obstante, aumenta la duración del tiempo de espera de las transacciones.

Por tanto, parece que el cumplimiento de la atomicidad de las transacciones debe llevar a una mayor probabilidad de esperas de larga duración o a crear la posibilidad de retrocesos en cascada.

El aislamiento de instantáneas, que se describe en la Sección 15.7, puede ser una solución parcial a este problema, así como el protocolo de *control de concurrencia optimista sin validación de lectura*, descrito en la Sección 15.9.3. Este último protocolo se diseñó específicamente para tratar con transacciones de larga duración que implican la interacción con usuarios. Aunque no garantiza la secuencialidad, el control de concurrencia optimista sin validación de lectura se usa extensamente.

Sin embargo, cuando las transacciones son de larga duración, es más probable que las actualizaciones en conflicto generen esperas o abortos adicionales. Estas consideraciones son la base de los conceptos alternativos de corrección de las ejecuciones concurrentes y de la recuperación de transacciones que se considerarán en el resto de esta sección.

26.6.2. Control de concurrencia

El objetivo fundamental del control de concurrencia de las bases de datos es asegurarse de que la ejecución concurrente de las transacciones no da lugar a una pérdida de la consistencia de la base de datos. El concepto de secuencialidad puede usarse para conseguir este objetivo, ya que todas las planificaciones secuenciables conservan la consistencia de las bases de datos. No obstante, no todas las planificaciones que conservan la consistencia de las bases de datos son secuenciables. Por ejemplo, considere nuevamente una base de datos bancaria que consista en dos cuentas, A y B, con el requisito de consistencia de que se conserve la suma $A + B$. Aunque la planificación de la Figura 26.5 no es secuenciable para conflictos, pese a todo, conserva la suma de $A + B$. También muestra dos aspectos importantes del concepto de corrección sin secuencialidad.

- La corrección depende de las restricciones de consistencia concretas para la base de datos.
- La corrección depende de las propiedades de las operaciones llevadas a cabo por cada transacción.

En general, no es posible llevar a cabo un análisis automático de las operaciones de bajo nivel de las transacciones y comprobar su efecto en las restricciones de consistencia de la base de datos. Sin embargo, hay técnicas más sencillas. Una de ellas es el uso de las restricciones de consistencia de la base de datos para dividir esta en subclases de datos en las que se puede administrar por separado

la concurrencia. Otra es el intento de tratar algunas operaciones distintas de **leer** y de **escribir** como operaciones fundamentales de bajo nivel y ampliar el control de concurrencia para trabajar con ellas.

Las notas bibliográficas hacen referencia a otras técnicas para asegurar la consistencia sin exigir secuencialidad. Muchas de estas técnicas aprovechan variedades del control de concurrencia multiversión (consulte la Sección 15.6). A las aplicaciones de procesamiento de datos más antiguas que solo necesitan una versión los protocolos multiversión les imponen una elevada sobrecarga de espacio para almacenar las versiones adicionales. Dado que muchas de las nuevas aplicaciones de bases de datos exigen el mantenimiento de las versiones de los datos, las técnicas de control de concurrencia que aprovechan varias versiones resultan prácticas.

T_1	T_2
leer(A)	
$A := A - 50$	
escribir(A)	
	leer(B)
	$B := B - 10$
	escribir(B)
leer(B)	
$B := B - 50$	
escribir(B)	
	leer(A)
	$A := A + 10$
	escribir(A)

Figura 26.5. Planificación no secuenciable para conflictos.

26.6.3. Transacciones anidadas y multinivel

Las transacciones de larga duración pueden considerarse como conjuntos de subtareas relacionadas o subtransacciones. Al estructurar cada transacción como un conjunto de subtransacciones, se puede mejorar el paralelismo, ya que puede que sea posible ejecutar en paralelo varias subtransacciones. Además, es posible trabajar con los fallos de las subtransacciones (debidos a abortos, fallos del sistema, etc.) sin tener que hacer retroceder toda la transacción de larga duración.

Una transacción anidada o multinivel T consiste en un conjunto $T = \{t_1, t_2,\ldots t_n\}$ de subtransacciones y en un orden parcial P sobre T. Cada subtransacción t_i de T puede abortar sin obligar a que T aborte. En su lugar, puede que T reinicie t_i o simplemente escoja no ejecutar t_i. Si se compromete t_i, esa acción no hace que t_i sea permanente (a diferencia de la situación del Capítulo 16). En vez de eso, t_i *se compromete con* T, y puede que todavía aborte (o exija compensación, consulte la Sección 26.6.4) si T aborta. La ejecución de T no debe violar el orden parcial P. Es decir, si aparece una arista $t_i \rightarrow t_j$ en el grafo de precedencia, $t_i \rightarrow t_j$ no debe estar en el cierre transitivo de P.

El anidamiento puede tener varios niveles de profundidad, representando la subdivisión de una transacción en subtareas, subsubtareas, etc. En el nivel inferior de anidamiento se tienen las operaciones estándar de bases de datos **leer** y **escribir** que se han usado anteriormente.

Si se permite a una subtransacción de T liberar los bloqueos al completarse, T se denomina **transacción multinivel.** Cuando una transacción multinivel representa una actividad de larga duración, a veces se denomina **saga.** De manera alternativa, si los bloqueos mantenidos por la subtransacción t_i de T se asignan de manera automática a T al concluir t_i, T se denomina **transacción anidada.**

Aunque el principal valor práctico de las transacciones multinivel surja en las transacciones complejas de larga duración, se usará el sencillo ejemplo de la Figura 26.5 para mostrar el modo en que el anidamiento puede crear operaciones de nivel superior que pueden mejorar la concurrencia. Se reescribe la transacción T_1 mediante las subtransacciones $T_{1,1}$ y $T_{1,2}$, que llevan a cabo operaciones de suma o de resta:

- T_1 consta de:
  - $T_{1,1}$, que resta 50 de A.
  - $T_{1,2}$, que suma 50 a B.

De manera parecida, se reescribe la transacción T_2 mediante las subtransacciones $T_{2,1}$ y $T_{2,2}$, que también llevan a cabo operaciones de suma o de resta:

- T_2 consta de:
  - $T_{2,1}$, que resta 10 de B.
  - $T_{2,2}$, que suma 10 a A.

No se especifica ninguna ordenación para $T_{1,1}$, $T_{1,2}$, $T_{2,1}$ ni $T_{2,2}$. Cualquier ejecución de estas subtransacciones generará un resultado correcto. La planificación de la Figura 26.5 corresponde a la planificación $<T_{1,1}, T_{2,1}, T_{1,2}, T_{2,2}>$.

26.6.4. Transacciones compensadoras

Para reducir la frecuencia de las esperas de larga duración, se preparan las actualizaciones no comprometidas para que se muestren a otras transacciones ejecutándose concurrentemente. En realidad, las transacciones multinivel pueden permitir esta exposición. No obstante, la exposición de datos no comprometidos crea la posibilidad de retrocesos en cascada. El concepto de **transacciones compensadoras** ayuda a tratar este problema.

Sea T la transacción que se divide en varias subtransacciones $t_1, t_2,\ldots t_n$. Una vez comprometida la subtransacción t_i, libera sus bloqueos. Ahora, si hay que abortar la transacción del nivel externo T, hay que deshacer el efecto de sus subtransacciones. Suponga que las subtransacciones $t_1,\ldots t_k$ se han comprometido y que t_{k+1} se estaba ejecutando cuando se tomó la decisión de abortar. Se pueden deshacer los efectos de t_{k+1} abortando esa subtransacción. Sin embargo, no es posible abortar las subtransacciones $t_1,\ldots t_k$, puesto que ya se han comprometido.

En su lugar, se ejecuta una nueva subtransacción tc_i, denominada *transacción compensadora*, para deshacer el efecto de cada subtransacción t_i. Es necesario que cada subtransacción t_i tenga su transacción compensadora tc_i. Las transacciones compensadoras deben ejecutarse en el orden inverso $tc_k,\ldots tc_1$. A continuación se ofrecen varios ejemplos de compensaciones:

- Considere la planificación de la Figura 26.5, que se ha demostrado correcta, aunque no secuenciable en cuanto a conflictos. Cada subtransacción libera sus bloqueos una vez se completa. Suponga que T_2 falla justo antes de su terminación, una vez que $T_{2,2}$ ha liberado sus bloqueos. Se ejecuta una transacción compensadora para $T_{2,2}$ que resta 10 de A y una transacción compensadora para $T_{2,1}$ que suma 10 a B.
- Considere una inserción en la base de datos por la transacción T_i que, como efecto lateral, provoca que se actualice el índice del árbol B$^+$. La operación de inserción puede haber modificado varios nodos del índice del árbol B$^+$. Otras transacciones pueden haber leído estos nodos al tener acceso a datos diferentes del registro insertado por T_i. Como en la Sección 16.7, se puede deshacer la inserción eliminando el registro insertado por T_i. El resultado es un árbol B$^+$ correcto y consistente, pero no necesariamente con exactamente la misma estructura que la que se tenía antes de que se iniciara T_i. Por tanto, la eliminación es una acción compensadora para la inserción.

- Considere una transacción de larga duración T_i que represente una reserva de viaje. La transacción T tiene tres subtransacciones: $T_{i,1}$, que hace las reservas de billetes de avión; $T_{i,2}$, que reserva los coches de alquiler, y $T_{i,3}$, que reserva las habitaciones de hotel. Suponga que el hotel cancela la reserva. En lugar de deshacer todas las T_i, se compensa el fallo de $T_{i,3}$ eliminando la reserva de hotel antigua y realizando una nueva.

Si el sistema falla en medio de la ejecución de una transacción del nivel externo, hay que hacer retroceder sus subtransacciones cuando se recupere. Las técnicas descritas en la Sección 16.7 se pueden usar con este fin.

La compensación del fallo de una transacción exige que se utilice la semántica de la transacción que ha fallado. Para determinadas operaciones, como el incremento o la inserción en un árbol B^+, la compensación correspondiente se define con facilidad. Para transacciones más complejas puede que los planificadores de la aplicación tengan que definir la forma correcta de compensación en el momento en que se codifique la transacción. Para las transacciones interactivas complejas puede que sea necesario que el sistema interactúe con el usuario para determinar la forma adecuada de compensación.

26.6.5. Problemas de implementación

Los conceptos sobre las transacciones estudiados en este apartado crean serias dificultades para su implementación. Aquí se presentan unas cuantas y se estudia el modo de abordarlas.

Las transacciones de larga duración deben sobrevivir a los fallos del sistema. Se puede asegurar que lo harán llevando a cabo una operación **rehacer** con las subtransacciones comprometidas, y una operación **deshacer** o una compensación para cualquier subtransacción de corta duración que estuviera activa en el momento del fallo. Sin embargo, estas acciones solo resuelven parte del problema. En los sistemas típicos de bases de datos los datos internos del sistema como las tablas de bloqueos y las marcas temporales de las transacciones se conservan en almacenamiento volátil. Para que se pueda reanudar una transacción de larga duración tras un fallo hay que restaurar esos datos. Por tanto, es necesario registrar no solo las modificaciones de la base de datos, sino también las modificaciones de los datos internos del sistema correspondientes a las transacciones de larga duración.

El registro histórico de las actualizaciones se hace más complicado cuando hay en la base de datos ciertos tipos de elementos de datos. Un elemento de datos puede ser un diseño CAD, el texto de un documento u otra forma de diseño compuesto. Estos elementos de datos son de gran tamaño físico. Por tanto, guardar los valores antiguos y nuevos del elemento de datos en un registro del registro histórico no resulta deseable.

Hay dos enfoques para reducir la sobrecarga de asegurar la recuperabilidad de elementos de datos de gran tamaño:

- **Registro histórico de operaciones.** Solo se guardan en el registro histórico la operación llevada a cabo en el elemento de datos y el nombre del elemento de datos. El registro histórico de operaciones también se denomina **registro histórico lógico.** Para cada operación debe haber una operación inversa. Se lleva a cabo la operación **deshacer** usando la operación inversa y la operación **rehacer** usando la misma operación. La recuperación mediante el registro histórico de operaciones resulta más difícil, ya que **rehacer** y **deshacer** no son idempotentes. Además, el uso del registro lógico para una operación que actualice varias páginas resulta muy complicado debido al hecho de que algunas, pero no todas, las páginas actualizadas pueden haberse escrito en el disco, por lo que resulta difícil aplicar tanto **rehacer** como **deshacer** a la operación en la imagen del disco durante la recuperación. El uso del registro histórico físico de rehacer y el registro histórico lógico de deshacer, tal y como se describen en la Sección 16.7, proporciona las ventajas de concurrencia del registro histórico lógico y evita los inconvenientes mencionados.
- **Registro histórico y paginación en la sombra.** El registro se usa para las modificaciones de elementos de datos de pequeño tamaño, pero los elementos de datos de gran tamaño a menudo se hacen recuperables mediante una técnica de paginación en la sombra o copia-en-escritura. Cuando se usa la paginación en la sombra es posible reducir la sobrecarga manteniendo solo copias de las páginas que realmente se modifiquen.

Independientemente de la técnica usada, las complejidades introducidas por las transacciones de larga duración y los elementos de datos de gran tamaño complican el proceso de recuperación. Por tanto, es deseable permitir que algunos datos no esenciales queden exentos del registro, y confiar en las copias de seguridad fuera de línea y en la intervención de las personas.

26.7. Resumen

- Los flujos de trabajo son actividades que implican la ejecución coordinada de varias tareas llevadas a cabo por diferentes entidades de proceso. Existen en las aplicaciones informáticas y en casi todas las actividades de una organización. Con el auge de las redes y la existencia de numerosos sistemas autónomos de bases de datos, los flujos de trabajo ofrecen una manera adecuada de llevar a cabo las tareas que implican a varios sistemas.
- Aunque los requisitos transaccionales ACID habituales resultan demasiado estrictos o no pueden implementarse para estas aplicaciones de flujo de trabajo, estos deben satisfacer un conjunto limitado de propiedades transaccionales que garanticen que no se dejen los procesos en estados inconsistentes.
- Los monitores de procesamiento de transacciones se desarrollaron inicialmente como servidores con varias hebras que podían atender a un gran número de terminales desde un único proceso. Desde entonces han evolucionado y hoy en día ofrecen la infraestructura necesaria para la creación y gestión de sistemas complejos de procesamiento de transacciones que tienen gran número de clientes y varios servidores. Proporcionan servicios como colas duraderas de las solicitudes de los clientes y de las respuestas de los servidores, el encaminamiento de los mensajes de los clientes a los servidores, la mensajería persistente, el equilibrio de carga y la coordinación del compromiso de dos fases cuando las transacciones tienen acceso a varios servidores.
- Los sistemas de comercio electrónico se han convertido en una parte esencial del comercio. Hay varios aspectos de las bases de datos en los sistemas de comercio electrónico. La administración de los catálogos, especialmente su personalización, se realiza con bases de datos. Los mercados electrónicos ayudan a establecer el precio de los productos mediante subastas, subastas inversas o bolsas. Se necesitan sistemas de bases de datos de alto rendimiento para manejar este intercambio. Los pedidos se liquidan mediante sistemas de pago electrónico, que también necesitan sistemas de bases de datos de alto rendimiento para manejar tasas de transacciones muy elevadas.
- Las memorias principales de gran tamaño se aprovechan en determinados sistemas para conseguir una gran productividad del sistema. En esos sistemas, el registro histórico constituye

un cuello de botella. Bajo el concepto de compromiso en grupo se puede reducir el número de salidas hacia el almacenamiento estable, lo que libera este cuello de botella.

- La gestión eficiente de las transacciones interactivas de larga duración resulta más compleja debido a las esperas de larga duración y a la posibilidad de los abortos. Dado que las técnicas de control de concurrencia usadas en el Capítulo 15 emplean las esperas, los abortos o las dos cosas, hay que considerar técnicas alternativas. Estas técnicas deben asegurar la corrección sin exigir la secuencialidad.
- Las transacciones de larga duración se representan como transacciones atómicas con operaciones atómicas de la base de datos anidadas en el nivel inferior. Si una de ellas falla, solo se abortan las transacciones activas de corta duración. Las transacciones activas de larga duración se reanudan una vez que se han recuperado todas las de corta duración. Si falla la transacción del nivel exterior se necesita una transacción compensadora para deshacer las actualizaciones de las transacciones anidadas que se hayan comprometido.
- En los sistemas con restricciones de tiempo real, la corrección de la ejecución no solo implica la consistencia de la base de datos, sino también el cumplimiento de los plazos. La amplia variabilidad de tiempos de ejecución de las operaciones de lectura y de escritura complica el problema de la gestión de las transacciones en sistemas con restricciones temporales.

Términos de repaso

- Monitor TP.
- Arquitecturas de los monitores TP.
  - Proceso por cliente.
  - Servidor único.
  - Varios servidores, un encaminador.
  - Varios servidores, varios encaminadores.
- Multitarea.
- Cambio de contexto.
- Servidor con varias hebras.
- Gestor de la cola.
- Coordinación de aplicaciones.
  - Gestor de recursos.
  - Llamada a procedimiento remoto (Remote Procedure Call, RPC).
- Flujos de trabajo transaccionales.
  - Tarea.
  - Entidad de procesamiento.
  - Especificación del flujo de trabajo.
  - Ejecución del flujo de trabajo.
- Estado del flujo de trabajo.
  - Estados de ejecución.
  - Valores de salida.
  - Variables externas.
- Atomicidad ante fallos del flujo de trabajo.
- Estados de terminación del flujo de trabajo.
  - Aceptable.
  - No aceptable.
  - Comprometido.
  - Abortado.
- Recuperación en los flujos de trabajo.
- Sistema gestor de flujos de trabajo.
- Arquitecturas de los sistemas gestores del flujo de trabajo.
  - Centralizadas.
  - Parcialmente distribuidas.
  - Completamente distribuidas.
- Gestión de procesos de negocio.
- Orquestación.
- Comercio electrónico.
- Catálogos electrónicos.
- Mercados.
  - Subastas.
  - Subastas inversas.
  - Bolsas.
- Liquidación de pedidos.
- Certificados digitales.
- Bases de datos en memoria principal.
- Compromiso en grupo.
- Sistemas de tiempo real.
- Plazos.
  - Estricto.
  - Firme.
  - Flexible.
- Bases de datos de tiempo real.
- Transacciones de larga duración.
- Exposición de datos no comprometidos.
- Ejecuciones no secuenciables.
- Transacciones anidadas.
- Transacciones multinivel.
- Saga.
- Transacciones compensadoras.
- Registro histórico lógico.

Ejercicios prácticos

26.1. Los sistemas de flujo de trabajo necesitan la gestión de la concurrencia y de la recuperación. Indique tres motivos por los que no se puede aplicar simplemente un sistema de bases de datos relacionales usando el bloqueo de dos fases, el registro histórico de operaciones físicas de deshacer y el compromiso de dos fases.

26.2. Considere un sistema de bases de datos en memoria principal que se recupera de un fallo del sistema. Explique las ventajas relativas de:

- Volver a cargar toda la base de datos en la memoria principal antes de reanudar el procesamiento de las transacciones.
- Cargar los datos a medida que los soliciten las transacciones.

26.3. Indique si un sistema de transacciones de alto rendimiento es necesariamente un sistema de tiempo real. Justifique la respuesta.

26.4. Explique el motivo por el que puede que no resulte práctico exigir la secuencialidad para las transacciones de larga duración.

26.5. Considere un proceso con varias hebras que entrega mensajes desde una cola duradera de mensajes persistentes. Pueden ejecutarse de manera concurrente diferentes hebras, que intentan entregar mensajes diversos. En caso de fallo en la entrega el mensaje debe restaurarse en la cola. Modele las acciones que lleva a cabo cada hebra como una transacción multinivel, de manera que no haga falta mantener los bloqueos en la cola hasta que se entregue cada mensaje.

26.6. Describa las modificaciones que hay que hacer en cada uno de los esquemas de recuperación tratados en el Capítulo 16 si se permiten las transacciones anidadas.

Explique también las diferencias que se producen si se permiten las transacciones multinivel.

Ejercicios

26.7. Explique cómo los monitores TP administran los recursos de la memoria y del procesador de manera más efectiva que los sistemas operativos habituales.

26.8. Compare las características de los monitores TP con las proporcionadas por los servidores web que soportan servlets (estos servidores se han denominado TP ligeros).

26.9. Considere el proceso de admisión de nuevos estudiantes en la universidad (o de nuevos empleados en la organización).

a. Describa una imagen de alto nivel del flujo de trabajo comenzando por el procedimiento de matrícula de los estudiantes.

b. Indique los estados de terminación aceptables y los pasos que implican la intervención de personas.

c. Indique los posibles errores (incluido el vencimiento del plazo) y el modo en que se tratan.

d. Estudie la cantidad de flujo de trabajo que se ha automatizado en la universidad.

26.10. Conteste las preguntas siguientes con respecto a sistemas electrónicos de pago.

a. Explique por qué las transacciones electrónicas realizadas usando números de la tarjeta de crédito pueden ser inseguras.

b. Un alternativa es tener una pasarela para el pago electrónico mantenida por la compañía de la tarjeta de crédito, y que el sitio que reciba el pago redirija a los clientes a la pasarela para efectuar este.

i Explique las ventajas que ofrece este sistema si la entrada no autentica al usuario.

ii Explique las ventajas si la pasarela tiene un mecanismo para autenticar al usuario.

c. Algunas compañías de tarjetas de crédito ofrecen un número de la tarjeta de un solo uso como método más seguro de pago electrónico. Los clientes se conectan con el sitio web de la compañía de tarjetas de crédito para conseguir un número de un solo uso. Explique la ventaja que ofrece este sistema con respecto a usar números de tarjeta de crédito normales. Explique también sus ventajas e inconvenientes con respecto a las pasarelas de pago con autentificación.

d. ¿Alguno de estos sistemas garantiza la misma privacidad que cuando los pagos se hacen en efectivo?

Justifique la respuesta.

26.11. Si toda la base de datos cabe en la memoria principal, indique si sigue haciendo falta un sistema de bases de datos para administrar los datos. Justifique la respuesta.

26.12. En la técnica de compromiso en grupo indique el número de transacciones que deben formar parte de un grupo.

Justifique la respuesta.

26.13. En un sistema de bases de datos que utilice el registro histórico de escritura adelantada indique el número de accesos a disco necesarios para leer un elemento de datos de una página de disco especificada en el peor caso posible.

Explique el motivo por el que esto supone un problema para los diseñadores de sistemas de bases de datos de tiempo real. (Sugerencia: suponga el caso cuando la memoria intermedia del disco está llena).

26.14. Indique la finalidad de las transacciones compensadoras. Presente dos ejemplos de su uso.

26.15. Explique las semejanzas entre un flujo de trabajo y una transacción de larga duración.

Notas bibliográficas

Gray y Reuter [1993] ofrecen una descripción detallada (y excelente) de los sistemas de procesamiento de transacciones, incluidos capítulos sobre los monitores TP. X/Open [1991] define la interfaz X/Open XA.

Fischer [2001] es autor de un manual sobre los sistemas de flujos de trabajo publicado en asociación con la Coalición para la Gestión de Flujos de Trabajo (Workflow Management Coalition). El sitio web de la coalición es www.wfmc.org. La descripción de flujos de trabajo que se ha dado sigue el modelo de Rusinkiewicz y Sheth [1995].

Loeb [1998] proporciona una descripción detallada de las transacciones electrónicas seguras.

García-Molina y Salem [1992] ofrecen una introducción a las bases de datos en memoria principal. Jagadish et ál. [1993] describen un algoritmo de recuperación diseñado para las bases de datos en memoria principal. En Jagadish et ál. [1994] se describe un gestor de almacenamiento para las bases de datos en memoria principal.

Las bases de datos de tiempo real se estudian en Lam y Kuo [2001]. El control de concurrencia y las planificaciones en las bases de datos de tiempo real se estudian en Haritsa et ál. [1990], Hong et ál. [1993] y Pang et ál. [1995]. Ozsoyoglu y Snodgrass [1995] es una recopilación de la investigación en las bases de datos de tiempo real y en las bases de datos temporales.

Las transacciones anidadas y las transacciones multinivel se presentan en Moss [1985], Lynch y Merritt [1986], Moss [1987], Haerder y Rothermel [1987], Rothermel y Mohan [1989], Weikum et ál. [1990], Korth y Speegle [1990], Weikum [1991], y Korth y Speegle [1994]. Los aspectos teóricos de las transacciones multinivel se presentan en Lynch et ál. [1988]. El concepto de saga se introdujo en García-Molina y Salem [1987].

Parte 9
Estudio de casos

Esta parte describe la forma en que los distintos sistemas de bases de datos integran los conceptos descritos anteriormente en el libro. Comienza en el Capítulo 27 con el estudio de un sistema de bases de datos de código abierto ampliamente usado: PostgreSQL. En los Capítulos 28, 29 y 30 se estudian tres sistemas comerciales de bases de datos muy usados: DB2 de IBM, Oracle y SQL Server de Microsoft. Representan tres de los sistemas comerciales de bases de datos más usados.

Cada uno de estos capítulos muestra características únicas de cada sistema de bases de datos: herramientas, variaciones y extensiones de SQL, y la arquitectura del sistema, incluyendo organización del almacenamiento, procesamiento de consultas, control de concurrencia, recuperación y réplicas. Los capítulos tratan solamente los aspectos clave de los productos de bases de datos que describen y por tanto no se deberían usar como una descripción completa. Además, puesto que los productos se mejoran periódicamente, sus detalles pueden cambiar. Cuando se usa una versión particular del producto hay que asegurarse de consultar en los manuales de usuario los detalles específicos.

Se debe tener presente que en los capítulos de esta parte se usa a menudo terminología industrial en lugar de académica. Por ejemplo, se usa *tabla* en lugar de *relación*, *fila* en lugar de *tupla* y *columna* en lugar de *atributo*.

PostgreSQL

27

Anastasia Ailamaki, Sailesh Krishnamurthy,
Spiros Papadimitriou, Bianca Schroeder,
Karl Schnaitter y Gavin Sherry

PostgreSQL es un sistema gestor de bases de datos relacionales de objetos de código abierto. Es un descendiente de uno de los primeros sistemas de este tipo, el sistema POSTGRES, desarrollado bajo la dirección del profesor Michael Stonebraker en la Universidad de California, en Berkeley. El nombre «postgres» proviene del nombre de un sistema pionero de bases de datos relacionales, Ingres, también desarrollado bajo la dirección de Stonebraker en Berkeley. Actualmente, PostgreSQL soporta muchos aspectos de SQL:2003 y ofrece características como las consultas complejas, las claves externas, los disparadores, las vistas, la integridad transaccional, la búsqueda de texto completo y la replicación de datos limitada. Además, los usuarios pueden ampliar PostgreSQL con nuevos tipos de datos, funciones, operadores y métodos de indexación. PostgreSQL trabaja con gran variedad de lenguajes de programación (incluidos C, C++, Java, Perl, TcI y Python), así como con las interfaces de bases de datos JDBC y ODBC. Otro punto fuerte de PostgreSQL es que, junto con MySQL, son los dos sistemas de bases de datos relacionales de código abierto más utilizados. La licencia de PostgreSQL es la licencia BSD, que concede permiso para el uso, modificación y distribución del código y de la documentación con cualquier propósito, libre de cargo.

27.1. Introducción

A lo largo de más de dos décadas, PostgreSQL ha tenido varias versiones importantes. El primer sistema prototipo, con el nombre de POSTGRES, se presentó en la conferencia SIGMOD de la ACM en 1988. La primera versión se distribuyó a los usuarios en 1989, proporcionando características como tipos de datos extensibles, un sistema de reglas preliminar y un lenguaje de consultas denominado POSTQUEL. Después de que las versiones posteriores añadieran un nuevo sistema de reglas, soporte para varios gestores de almacenamiento y un ejecutor de consultas mejorado, los desarrolladores del sistema se centraron en la portabilidad y en el rendimiento hasta 1994, cuando se añadió un intérprete del lenguaje SQL. Con un nuevo nombre, Postgres95, el sistema se distribuyó por web y, posteriormente, fue comercializado por Illustra Information Technologies (que posteriormente se fusionó con Informix, que ahora es propiedad de IBM). Hacia 1996, el nombre Postgres95 fue sustituido por PostgreSQL, para reflejar la relación entre el POSTGRES original y las versiones más recientes con capacidades de SQL.

PostgreSQL se puede ejecutar bajo prácticamente todos los sistemas operativos tipo Unix, incluidos Linux y OS X para Apple, de Macintosh. PostgreSQL también puede ejecutarse bajo Microsoft Windows en el entorno Cygwin, el cual proporciona emulación de Linux bajo Windows. La versión más reciente, la 8.0, publicada en enero de 2005, ofrece soporte nativo de Microsoft Windows.

Hoy en día PostgreSQL se utiliza para implementar varias aplicaciones de investigación y de producción diferentes (como el sistema de información geográfica PostGIS) y como herramienta educativa en muchas universidades. El sistema sigue evolucionando gracias a las contribuciones de una comunidad de alrededor de 1.000 desarrolladores. En este capítulo se explicará el funcionamiento de PostgreSQL, comenzando por las interfaces y los lenguajes de usuario y siguiendo por el corazón del sistema (las estructuras de datos y los mecanismos de control de concurrencia).

27.2. Interfaces de usuario

La distribución estándar de PostgreSQL incluye herramientas de línea de comandos para la administración de las bases de datos. Sin embargo, existe un gran número de herramientas de administración y de diseño, comerciales y de código abierto, que soportan PostgreSQL. Los desarrolladores de software también puede utilizar PostgreSQL utilizando un amplio conjunto de interfaces de programación.

27.2.1. Interfaces de terminal interactivas

Como la mayoría de los sistemas de bases de datos, PostgreSQL ofrece herramientas de línea de comandos para la administración de bases de datos. El principal cliente de terminal interactiva es psql, que se modeló a partir del intérprete de comandos de Unix y permite la ejecución de comandos de SQL en el servidor, así como otras operaciones (como la copia en el lado del cliente). Algunas de sus características son:

- **Variables.** psql ofrece características de sustitución de variables, parecidas a los intérpretes de comandos Unix habituales.
- **Interpolación SQL.** El usuario puede sustituir («interpolar») las variables de psql en instrucciones SQL regulares colocando un punto y coma delante del nombre de la variable.
- **Edición de la línea de comandos.** psql utiliza la biblioteca readline de GNU para hacer más cómoda la edición de líneas, que permite autocompletar las entradas mediante tabuladores.

También se puede acceder a PostgreSQL desde los intérpretes de comandos tipo Tk y Tcl, que proporcionan un lenguaje de comandos flexible para prototipado rápido. Esta función se activa en Tcl/Tk cargando la biblioteca pgtcl, que se distribuye como extensión opcional con PostgreSQL.

27.2.2. Interfaces gráficas

La distribución estándar de PostgreSQL no contiene ninguna herramienta gráfica. No obstante, existen varias herramientas con interfaz gráfica de usuario, de forma que se puede escoger entre las posibilidades comerciales y las de código abierto. Muchas de ellas experimentan ciclos de publicación rápidos; la lista siguiente refleja la situación en el momento de escribir este libro.

Existen herramientas gráficas para la administración, como pgAccess y pgAdmin, la última de las cuales puede verse en la Figura 27.1. Entre las herramientas para el diseño de bases de datos están TORA y Data Architect, la última de las cuales puede verse en la Figura 27.2. PostgreSQL trabaja con varias herramientas comerciales para el diseño de formularios y la generación de informes. Entre las posibilidades de código abierto se encuentran Rekall (que se muestra en las Figuras 27.3 y 27.4), GNU Report Generator y un conjunto de herramientas más general, GNU Enterprise.

27.2.3. Interfaces para lenguajes de programación

PostgreSQL ofrece interfaces nativas para ODBC y para JDBC, así como enlaces con la mayor parte de los lenguajes de programación, incluidos C, C++, PHP, Perl, Tcl/Tk, ECPG, Python y Ruby.

La interfaz de programación para aplicaciones en C de PostgreSQL es libpq, que también es el motor subyacente de la mayor parte de los enlaces de lenguajes de programación (por tanto, todas sus características también están disponibles en los demás lenguajes soportados). La biblioteca libpq soporta tanto la ejecución síncrona de los comandos de SQL y de las instrucciones preparadas, como la asíncrona y las sentencias preparadas mediante una interfaz segura para hebras y reentrante. Los parámetros de conexión de libpq se pueden configurar de manera muy flexible, para establecer variables de entorno, establecer la configuración en archivos locales o crear entradas en un servidor LDAP.

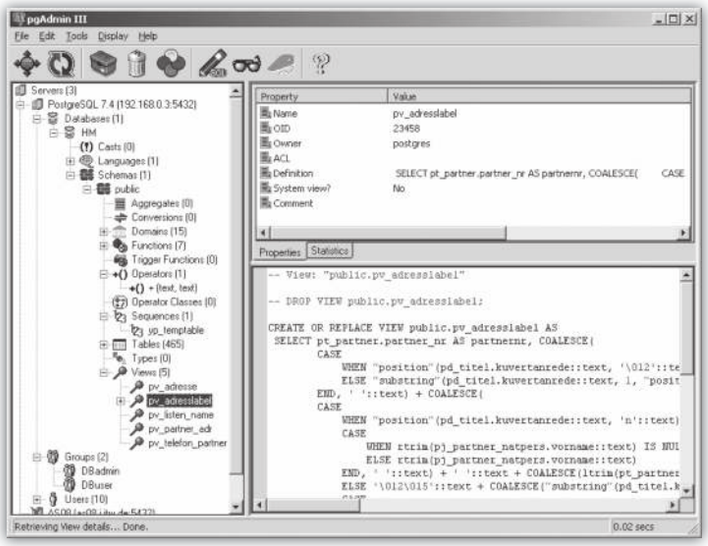


Figura 27.1. pgAdmin III: una interfaz gráfica de usuario de código abierto para la administración de bases de datos.

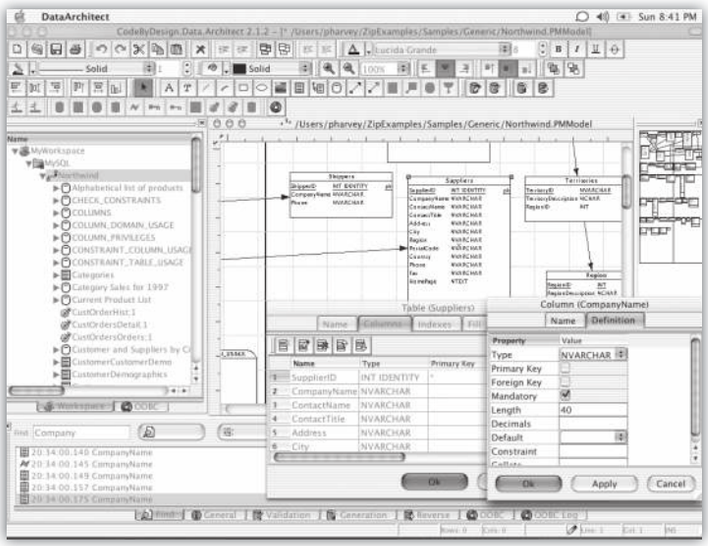


Figura 27.2. Data Architect: una interfaz gráfica de usuario multiplataforma para el diseño de bases de datos.

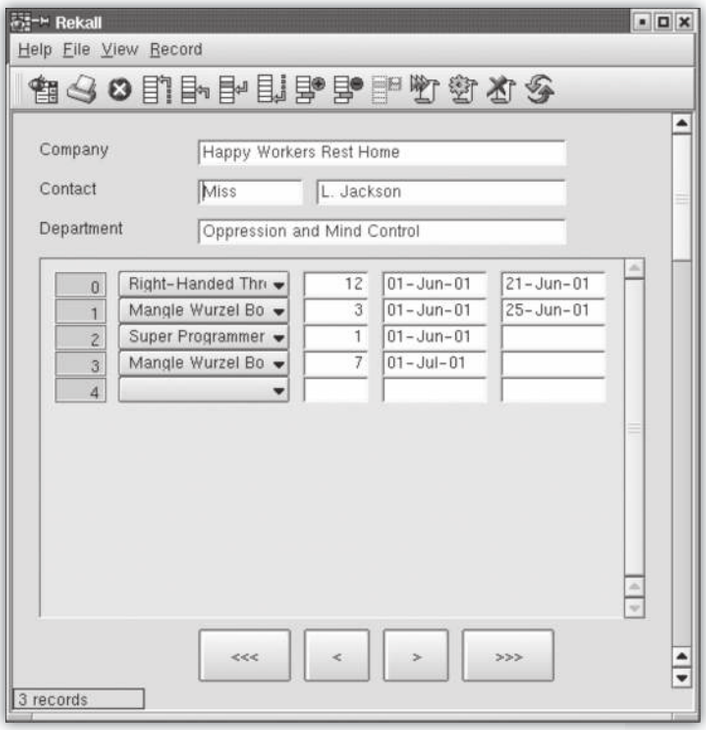


Figura 27.3. Rekall: interfaz gráfica de usuario para diseño de formularios.

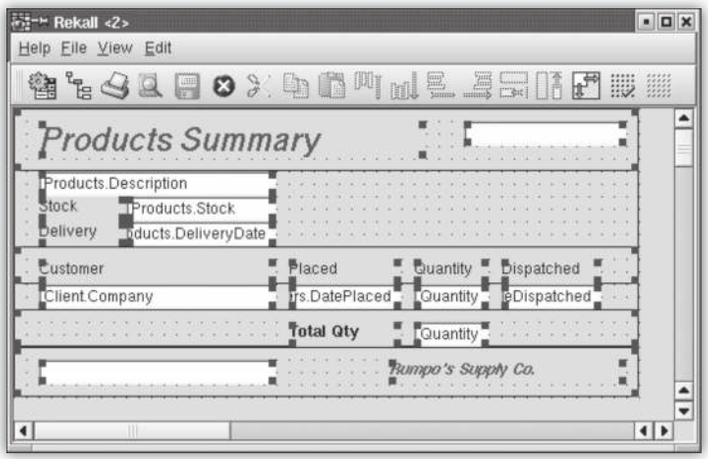


Figura 27.4. Rekall: interfaz gráfica de usuario para diseño de informes.

27.3. Variaciones y extensiones de SQL

La versión actual de PostgreSQL cumple casi todas las características del nivel básico de SQL-92, así como muchas de las características de los niveles intermedio y completo. También soporta muchas de las características de SQL:1999 y SQL:2003, incluyendo la mayoría de las características de objetos-relacionales descritas en el Capítulo 22 y las características SQL/XML para datos XML descritos en el Capítulo 23. De hecho, algunas características de la norma actual de SQL (como los arrays, las funciones y la herencia) fueron pioneras en PostgreSQL o sus antecesores. Carece de características OLAP (principalmente, **cube** y **rollup**), pero los datos de PostgreSQL pueden cargarse con facilidad en servidores OLAP externos de código abierto (como Mondrian), así como en productos comerciales.

27.3.1. Tipos de PostgreSQL

PostgreSQL incluye soporte para varios tipos no normalizados, que resultan útiles para dominios de aplicaciones concretas. Además, los usuarios pueden definir nuevos tipos con el comando **create type.** Esto incluye nuevos tipos básicos de bajo nivel, generalmente escritos en C (consulte la Sección 27.3.3.1).

27.3.1.1. El sistema de tipos de PostgreSQL

Los tipos de PostgreSQL pueden clasificarse en las siguientes categorías:

- **Tipos básicos.** Los tipos básicos también se conocen como **tipos abstractos de datos;** es decir, módulos que encapsulan al mismo tiempo un estado y un conjunto de operaciones. Se implementan por debajo del nivel SQL, generalmente en un lenguaje como C (consulte la Sección 27.3.3.1). Entre los ejemplos se encuentran **int4** (que ya está incluido en PostgreSQL) y **complex** (que se incluye como tipo de extensión opcional). Un tipo básico puede representar o un valor escalar individual o un array de tamaño variable de valores. Para cada tipo escalar que exista en la base de datos PostgreSQL crea automáticamente un tipo array que puede almacenar valores de ese mismo tipo escalar.
- **Tipos compuestos.** Se corresponden con las filas de las tablas; es decir, son una lista de nombres de campos y de sus tipos respectivos. Siempre que se crea una tabla se crea implícitamente un tipo compuesto, aunque los usuarios también pueden crearlos de manera explícita.
- **Dominios.** Un tipo dominio se define juntando un tipo básico con una restricción que debe satisfacer los valores del tipo. Los valores del tipo de dominio y el tipo básico asociado se pueden intercambiar, siempre que se satisfaga la restricción. Un dominio también puede tener un valor predeterminado, cuyo significado es similar al valor predeterminado de una columna de la tabla.
- **Tipos enumerados.** Estos tipos son similares a los tipos enum de los lenguajes de programación como C o Java. Un tipo enumerado es esencialmente una lista fija de valores con nombre. En PostgreSQL los tipos enumerados se pueden convertir a la representación textual de su nombre, pero esta conversión hay que indicarla explícitamente en algunos casos para asegurar la seguridad de los tipos. Por ejemplo, no se pueden comparar los valores de diferentes tipos enumerados sin una conversión explícita a tipos compatibles.
- **Pseudotipos.** Actualmente, PostgreSQL soporta los pseudotipos siguientes: *any*, *anyarray*, *anyelement*, *anyenum*, *anynonarray*, *cstring*, *internal*, *opaque*, *language_handler*, *record*, *trigger* y *void*. No pueden utilizarse en los tipos compuestos (y, por tanto, no pueden utilizarse para las columnas de las tablas), pero pueden emplearse como argumentos y tipos de retorno de funciones definidas por los usuarios.
- **Tipos polimórficos.** Cuatro de los pseudotipos: *anyelement*, *anyarray*, *anynonarray* y *anyenum*, se conocen colectivamente como **polimórficos.** Las funciones con argumentos de estos tipos (denominadas, a su vez, **funciones polimórficas**) pueden operar sobre cualquier tipo. PostgreSQL tiene un esquema sencillo de resolución de tipos que exige que: (1) en cada invocación de una función polimórfica todas las ocurrencias de los tipos polimórficos estén vinculadas al mismo tipo (es decir, una función definida como *f* (*anyelement*, *anyelement)* solo puede operar sobre parejas del mismo tipo) y (2) si el tipo devuelto es polimórfico, al menos uno de los argumentos debe ser del mismo tipo polimórfico.

27.3.1.2. Tipos no estándar

Los tipos descritos en esta sección se incluyen en la distribución estándar. Además, gracias a la naturaleza abierta de PostgreSQL, se han aportado varios tipos adicionales, como los números complejos y el ISBN/ISSN (consulte la Sección 27.3.3).

Los tipos de datos geométricos (*point*, *line*, *lseg*, *box*, *polygon*, *path*, *circle*) se utilizan en los sistemas de información geográfica para representar objetos espaciales bidimensionales como los puntos, los segmentos de línea, los polígonos, los caminos y los círculos. En PostgreSQL se dispone de numerosas funciones y operadores para llevar a cabo diferentes operaciones geométricas como el cambio de escala, las traslaciones, las rotaciones y la determinación de las intersecciones. Además, PostgreSQL soporta el indexado de estos tipos mediante árboles R (Secciones 25.3.5.3 y 27.5.2.1).

La búsqueda de texto completo se realiza en PostgreSQL utilizando el tipo *tsvector*, que representa un documento, y el tipo *tsquery*, que representa una consulta de texto completo. Un *tsvector* almacena las palabras distintas de un documento, tras convertir las variantes de cada palabra a una forma normal común, por ejemplo, eliminando los radicales de la palabra. PostgreSQL proporciona funciones para convertir texto en bruto a un *tsvector* y concatenar documentos. Un *tsquery* especifica las palabras a buscar en un documento candidato, con múltiples palabras conectadas por operadores booleanos. Por ejemplo, la consulta ´index & !(tree | hash)´ busca documentos que contengan «index» sin usar las palabras «tree» ni «hash». PostgreSQL dispone nativamente de operaciones sobre tipos de texto completo, incluyendo características del lenguaje y búsqueda en índices.

PostgreSQL ofrece tipos de datos para el almacenamiento de direcciones de red. Estos tipos de datos permiten que las aplicaciones de administración de redes utilicen bases de datos de PostgreSQL como almacenes de datos. Para aquellos familiarizados con las redes de computadoras, a continuación se ofrece un breve resumen de esta característica. Hay tipos diferentes para direcciones IPv4, direcciones IPv6 direcciones de Control de Acceso a Medios (Media Access Control, MAC) (*cidr*, *inet* y *macaddr*, respectivamente). Tanto el tipo *inet* como el tipo *cidr* pueden almacenar direcciones IPv4 y direcciones IPv6, con máscaras de subred opcionales. Su principal diferencia estriba en el formato de entrada y de salida, así como en la restricción de que las direcciones de dominios de Internet sin clase (*classless internet domain routing*: CIDR) no acepten valores con bits diferentes de cero a la derecha de la máscara de subred. El tipo *macaddr* se utiliza para almacenar direcciones MAC (generalmente, direcciones de hardware de tarjetas Ethernet). PostgreSQL soporta el indexado y la ordenación de estos tipos, así como un conjunto de operaciones (incluidas la comprobación de subredes y la asignación de direcciones MAC a nombres de fabri-

cantes de hardware). Además, estos tipos ofrecen la comprobación de errores de entrada. Por tanto, son preferibles a los campos de texto plano.

El tipo *bit* de PostgreSQL puede almacenar cadenas de unos y ceros tanto de longitud fija como de longitud variable. PostgreSQL soporta los operadores lógicos de bits y las funciones de manipulación de cadenas de caracteres.

27.3.2. Reglas y otras características de las bases de datos activas

PostgreSQL soporta las restricciones y los disparadores de SQL (y los procedimientos almacenados; consulte la Sección 27.3.3). Además, ofrece reglas de reescritura de consultas que pueden declararse en el servidor.

PostgreSQL permite comprobar **(check)** las restricciones, las restricciones de existencia **(not null)** y las restricciones de clave principal y de clave externa (con borrados restrictivos y en cascada).

Al igual que otros muchos sistemas de bases de datos relacionales, PostgreSQL soporta los disparadores, que resultan útiles para las restricciones no triviales y para comprobar o hacer cumplir la consistencia. Las funciones disparadoras pueden escribirse en lenguajes procedimentales como PL/pgSQL (consulte la Sección 27.3.3.4) o en C, pero no en mero SQL. Los disparadores pueden ejecutarse antes o después de las operaciones **insert, update** o **delete** tanto una sola vez por fila modificada como una sola vez por instrucción SQL.

El sistema de reglas de PostgreSQL permite que los usuarios definan las reglas de reescritura de consultas en el servidor de bases de datos. A diferencia de los procedimientos almacenados y de los disparadores, el sistema de reglas interviene entre el analizador de consultas y el planificador, y modifica las consultas de acuerdo con el conjunto de reglas. Una vez transformado el árbol de la consulta original en uno o más árboles, estos se pasan al planificador de consultas. Por tanto, el planificador tiene toda la información necesaria (las tablas que hay que explorar, las relaciones entre ellas, las calificaciones, la información de reunión, etc.) y puede presentar un plan de ejecución eficiente, aunque intervengan reglas complejas.

La sintaxis general para la declaración de reglas es:

create rule *nombre_regla* **as**
  on {**select** | **insert** | **update** | **delete**}
  to *tabla* [**where** *calificación_regla*]
  do [**instead**] {**nothing** | *comando* | (*comando* ; *comando*...)}

El resto de esta sección ofrece ejemplos que ilustran las posibilidades del sistema de reglas. En la documentación de PostgreSQL se pueden encontrar más detalles sobre el modo en que las reglas se adaptan a los árboles de consultas y la manera en que estos últimos se transforman posteriormente (consulte las notas bibliográficas). El sistema de reglas se implementa en la fase de reescritura del procesamiento de las consultas y se explica en la Sección 27.6.1.

En primer lugar, PostgreSQL utiliza las reglas para implementar las vistas. La definición de una vista como:

create view *mivista* **as select** * **from** *mitabla*;

se transforma en la siguiente definición de regla:

create table *mivista* (*la misma lista de columnas que en mitabla*);
create rule *_retorno* **as on select to** *mivista* **do instead**
  select * **from** *mitabla*;

Las consultas a *mivista* se transforman antes de su ejecución en consultas a la tabla subyacente *mitabla*. La sintaxis **create view** se considera la mejor forma de programación en este caso, ya que es más concisa y también evita la creación de vistas que hagan referencia a otras (lo que es posible si las reglas se declaran de manera poco cuidadosa, y puede dar lugar a errores de confusión en los tiempos de ejecución). No obstante, las reglas pueden servir para definir de manera explícita acciones de actualización de vistas (las instrucciones **create view** no permiten esto).

Como ejemplo adicional, considere el caso de que el usuario desee registrar todos los incrementos de sueldo de los profesores. Se podría lograr mediante una regla como:

create rule *control_sueldos* **as on update to** *profesor*
  where new.*sueldo* <> **old**.*sueldo*
  do insert into *control_sueldos*
  values (current_timestamp, current_user,
    new.*nombre_emp*, **old**.*sueldo*, **new**.*sueldo*);

Finalmente, se ofrece una regla de inserción y actualización ligeramente más complicada. Suponga que se almacenan los aumentos salariales pendientes en la tabla *aumentos_salariales* (*nombre_emp*, *aumento*). Se puede declarar la tabla adicional *aumentos_aprobados* con los mismos campos y definir después la regla siguiente:

create rule *aumentos_aprobados_insertar*
  as on insert to *aumentos_aprobados*
  do instead
  update *profesor*
    set *sueldo* = *sueldo* + **new**.*aumento*
    where *nombre_emp* = **new**.*nombre_emp*;

Entonces la consulta siguiente:

insert into *aumentos_aprobados* **select** *
  from *aumentos_salariales*;

actualizará a la vez todos los sueldos de la tabla *profesor*. Como se especifica la palabra clave **instead** en la regla, la tabla *aumentos_aprobados* no se modifica.

Existe cierto solapamiento entre la funcionalidad de las reglas y los disparadores por filas. El sistema de reglas de PostgreSQL puede servir para implementar la mayor parte de los disparadores, pero algunos tipos de restricciones (especialmente las claves externas) no se pueden implementar mediante reglas. Además, los disparadores pueden generar mensajes de error para indicar el no cumplimiento de las restricciones, mientras que las reglas solo pueden forzar la integridad de datos suprimiendo, silenciosamente, los valores no válidos. Por otra parte, los disparadores no pueden usarse para las acciones **update** ni **delete** sobre vistas. Dado que no hay ningún dato real en las relaciones de vistas, nunca se llamaría al disparador.

Una diferencia importante entre disparadores y vistas es que los disparadores se activan una vez por cada fila afectada. Las reglas, por su parte, manipulan el árbol de consultas antes de planificar la consulta. Por tanto, si una sentencia afecta a muchas filas, las reglas son más eficientes que los disparadores.

La implementación de disparadores y restricciones en PostgreSQL se esboza brevemente en la Sección 27.6.4.

27.3.3. Extensibilidad

Al igual que la mayor parte de los sistemas de bases de datos relacionales, PostgreSQL almacena la información sobre las bases de datos, las tablas, las columnas, etc., en lo que suele denominarse **catálogos del sistema,** que se presentan ante el usuario como tablas normales. Otros sistemas de bases de datos relacionales se suelen ampliar modificando procedimientos incluidos en el código fuente o mediante la carga de módulos especiales de extensión escritos por el fabricante.

Sin embargo, a diferencia de la mayor parte de los sistemas de bases de datos relacionales, PostgreSQL va un paso más allá y almacena mucha más información en los catálogos: no solo la información sobre las tablas y las columnas, sino también la información sobre los tipos de datos, las funciones, los métodos de acceso, etc. Por tanto, resulta sencillo para los usuarios ampliar PostgreSQL, lo cual facilita la creación rápida de prototipos de nuevas aplicaciones y de estructuras de almacenamiento. PostgreSQL también puede incorporar en el servidor código escrito por los usuarios, mediante la carga dinámica de los objetos compartidos. Esto ofrece un enfoque alternativo a la escritura de extensiones que puede utilizarse cuando las extensiones basadas en los catálogos no resultan suficientes.

Además, el módulo contrib de la distribución de PostgreSQL incluye numerosas funciones de usuario (por ejemplo, iteradores de *arrays*, comparación difusa de cadenas de caracteres, funciones criptográficas), tipos básicos (por ejemplo, contraseñas cifradas, ISBN/ISSN, cubos *n*-dimensionales) y extensiones de los índices (por ejemplo, árboles RD, indexado de texto completo). Gracias a la naturaleza abierta de PostgreSQL hay una gran comunidad de profesionales y entusiastas de PostgreSQL que también amplían PostgreSQL prácticamente a diario. Los tipos de las extensiones son idénticos en funcionalidad a los tipos predefinidos (consulte también la Sección 27.3.1.2); los últimos simplemente se hallan ya enlazados en el servidor y están registrados previamente en el catálogo del sistema. De manera parecida, esta es la única diferencia entre las funciones intrínsecas y las ampliadas.

27.3.3.1. Tipos

PostgreSQL permite que los usuarios definan tipos compuestos, así como que amplíen los tipos básicos disponibles.

Las definiciones de los tipos compuestos son parecidas a las definiciones de las tablas (de hecho, la última lleva a cabo la primera de manera implícita). Los tipos compuestos independientes suelen resultar útiles para los argumentos de las funciones. Por ejemplo, la definición:

create type *t_ciudad* **as** (*nombre* **varchar**(80), *provincia* **char**(2));

permite que las funciones acepten y devuelvan las tuplas *t_ciudad*, aunque no haya ninguna tabla que contenga de manera explícita filas de este tipo.

Los tipos enumerados son fáciles de definir, simplemente indicando los nombres de los valores. En el siguiente ejemplo se crea un tipo enumerado que representa el estado de un producto de software.

create type *t_estado* **as enum** (´alpha´, ´beta´, ´release´);

El orden en que se indican los nombres es significativo en la comparación de valores de un tipo enumerado. Esto puede ser útil en una sentencia como:

select *nombre* **from** *productos*
where *estado* > ´alpha´

que devuelve todos los nombre de los productos que han pasado del estado alpha.

La adición de tipos básicos a PostgreSQL es directa; se puede encontrar un ejemplo en complex.sql y complex.c en los tutoriales de la distribución de PostgreSQL. Los tipos básicos pueden declararse en C, por ejemplo:

```
typedef struct Complejo {
   double x;
   double y;
} Complejo;
```

A continuación el usuario tiene que definir las funciones para leer y escribir los valores del nuevo tipo en formato de texto (consulte la Sección 27.3.3.2). Posteriormente, el nuevo tipo puede registrarse empleando la sentencia:

create type *complejo* {
internallength = 16,
input = *complejo_entrada*,
output = *complejo_salida*,
alignment = **double**
};

suponiendo que las funciones de E/S de texto se hayan registrado como *complejo_entrada* y *complejo_salida*.

El usuario también tiene la posibilidad de definir funciones de E/S binarias (para un volcado de datos más eficiente). Los tipos ampliados pueden utilizarse como los tipos básicos de PostgreSQL ya existentes. De hecho, la única diferencia es que los tipos ampliados se cargan y enlazan dinámicamente en el servidor. Además, los índices pueden ampliarse fácilmente para que manejen los nuevos tipos básicos; consulte la Sección 27.3.3.3.

27.3.3.2. Funciones

PostgreSQL permite que los usuarios definan funciones que se almacenen y ejecuten en el servidor. También soporta la sobrecarga de funciones (es decir, se pueden declarar funciones que utilicen el mismo nombre pero con argumentos de tipos diferentes). Las funciones se pueden escribir como instrucciones de SQL. Además, se soportan varios lenguajes procedimentales (se tratan en la Sección 27.3.3.4). Finalmente, PostgreSQL tiene una interfaz para programación de aplicaciones para añadir funciones escritas en C (que se explica en esta sección).

Las funciones definidas por los usuarios pueden escribirse en C (o en un lenguaje con convenios de llamada compatibles, como C++). Los convenios de codificación reales son básicamente los mismos para las funciones definidas por los usuarios que se cargan de manera dinámica y para las funciones internas (que se enlazan en el servidor de manera estática). Por tanto, la biblioteca de funciones interna estándar es una fuente abundante de ejemplos de codificación para las funciones de C definidas por los usuarios. Una vez creada la biblioteca compartida que contiene la función, se registra en el servidor una declaración como la siguiente:

create function *complejo_salida*(*complejo*)
returns cstring
as ´nombre_archivo_objeto_compartido´
language C immutable strict;

Se da por supuesto que el punto de entrada al archivo del objeto compartido es el nombre de la función de SQL (aquí, *complejo_salida*), a menos que se especifique lo contrario.

El ejemplo siguiente continúa al de la Sección 27.3.3.1. La interfaz de programación de aplicaciones oculta la mayor parte de los detalles internos de PostgreSQL. Por tanto, el código C para la función de salida de texto anterior de valores *complejo* es bastante sencillo:

```
PG_FUNCTION_INFO_V1(complejo_salida);
Datum complejo_salida(pg_function_args) {
   Complejo *complejo = (Complejo *) pg_getarg_pointer(0);
   char *resultado;
   resultado = (char *) palloc(100);
   snprintf(resultado, 100, "(%g, %g)", complejo->x,
            complejo->y);
   pg_return_cstring(resultado);
}
```

La primera línea declara la función complejo_salida, y las siguientes líneas implementan la función de salida. El código utiliza varios elementos específicos de PostgreSQL, como la función palloc, que asigna dinámicamente memoria controlada por un gestor de memoria de PostgreSQL. Puede encontrar más detalles en la documentación de PostgreSQL (consulte las notas bibliográficas).

Las funciones agregadas de PostgreSQL operan actualizando los **valores de estado** mediante funciones de **transición de estado** que se llaman para cada valor de la tupla del grupo de agregación. Por ejemplo, el estado del operador **avg** consta de la suma y el recuento de los valores. A medida que llega cada tupla, la función de transición simplemente debe añadir su valor a la suma y aumentar en uno el recuento. De manera opcional, se puede llamar a una función *final* para que calcule el valor de retorno de acuerdo con la información de estado. Por ejemplo, la función final para **avg** simplemente dividiría la suma ejecutada entre el recuento y devolvería el resultado.

Por tanto, la definición de un operador nuevo es tan sencilla como la definición de esas dos funciones. Para el ejemplo del tipo *complejo*, si *complejo_suma* es una función definida por los usuarios que toma dos argumentos complejos y devuelve su suma, entonces el operador de agregación **sum** puede ampliarse a los números complejos mediante la siguiente declaración:

create aggregate *sum* (
  sfunc = *suma_complejo*,
  basetype = *complejo*,
  stype = *complejo*,
  initcond = ´(0,0)´
);

Observe el empleo de la sobrecarga de funciones: PostgreSQL llamará a la función de agregación *sum* adecuada en función del tipo real del argumento que se invoque. El *basetype* es el tipo de argumento y *stype* es el tipo del valor de estado. En este caso, no hace falta ninguna función final, ya que el valor devuelto es el propio valor de estado (es decir, la suma total en ambos casos).

Las funciones definidas por los usuarios también pueden invocarse mediante sintaxis de operadores. Más allá del mero «azúcar sintáctico» para la invocación de funciones, las declaraciones de operadores pueden ofrecer sugerencias al optimizador de consultas para que mejore el rendimiento. Estas sugerencias pueden incluir información sobre la conmutatividad, la restricción y la estimación de selectividad de las uniones, así como otras propiedades relacionadas con los algoritmos de reunión.

27.3.3.3. Extensiones de los índices

PostgreSQL soporta los índices habituales de árbol B y asociativos, así como dos índices que son únicos de PostgreSQL: el árbol de búsqueda generalizado (*generalized search tree*: GiST) y el índice invertido generalizado (*generalized inverted index*: GIN), útil para la indexación de texto completo (estas estructuras se describen en la Sección 27.5.2.1). Finalmente, PostgreSQL proporciona indexación de objetos espaciales de dos dimensiones con índices de árbol R, que se implementan usando los índices GiST. Todos ellos pueden extenderse fácilmente para incluir nuevos tipos básicos.

La adición de extensiones de los índices para un tipo dado exige la definición de una **clase de operador** que encapsule lo siguiente:

- **Estrategias para métodos de indexado.** Se trata de un conjunto de operadores que pueden utilizarse como calificadores en las cláusulas **where.** El conjunto concreto depende del tipo de índice. Por ejemplo, los índices de árbol B pueden recuperar rangos de objetos, por lo que el conjunto consiste en cinco operadores ($<$, $<=$, $=$, $>=$ y $>$), que pueden aparecer en las cláusulas **where** que afecten a índices de árbol B. Los índices asociativos solo permiten comprobar las igualdades y los índices de árbol R permiten varias relaciones espaciales (por ejemplo: contiene, a la izquierda, etc.).
- **Rutinas de soporte de los métodos de indexado.** El conjunto anterior de operadores no suele ser suficiente para la operación del índice. Por ejemplo, los índices asociativos necesitan una función que calcule el valor asociativo de cada objeto. Los índices de árbol R necesitan poder calcular las intersecciones y las uniones, así como estimar el tamaño de los objetos indexados.

Por ejemplo, si se definen las funciones y operadores siguientes para comparar la magnitud de los números *complejos* (consulte la Sección 27.3.3.1), entonces se puede hacer que esos objetos sean indexables mediante la siguiente declaración:

create operator class *ops_abs_complejo*
  default for type *complejo* **using btree as**
    operator 1 <(*complejo*, *complejo*),
    operator 2 <= (*complejo*, *complejo*),
    operator 3 = (*complejo*, *complejo*),
    operator 4 >= (*complejo*, *complejo*),
    operator 5 > (*complejo*, *complejo*),
    function 1 *cmp_abs_complejo*(*complejo*, *complejo*);

Las instrucciones **operator** definen los métodos estratégicos y las sentencias **function** definen los de soporte.

27.3.3.4. Lenguajes procedimentales

Las funciones y los procedimientos almacenados pueden escribirse en varios lenguajes procedimentales. Además, PostgreSQL define una interfaz para programación de aplicaciones con objeto de aprovechar cualquier lenguaje de programación con esta finalidad. Los lenguajes de programación pueden registrarse a petición de los usuarios y pueden ser **dignos de confianza** (*trusted*) o **no dignos de confianza** (*untrusted*). Estos últimos permiten un acceso ilimitado al SGBD y al sistema de archivos, y escribir en ellos las funciones almacenadas exige privilegios de superusuario.

- PL/pgSQL: se trata de un lenguaje digno de confianza que añade a SQL posibilidades de programación procedimental (por ejemplo, variables y control de flujo). Es muy parecido a PL/SQL de Oracle. Aunque no se puede transferir el código tal cual entre uno y otro, la portabilidad suele resultar sencilla.
- **PL/Tcl**, **PL/Perl** y **PL/Python.** Estos lenguajes aprovechan la potencia de Tcl, Perl y Python para escribir en el servidor funciones y procedimientos almacenados. Los dos primeros están disponibles tanto en versión digna de confianza como en versión no digna de confianza (PL/Tcl, PL/Perl y PL/TclU, PL/PerlU, respectivamente), mientras que PL/Python no es digno de confianza en el momento de escribir este libro. Cada uno de ellos tiene enlaces que permiten el acceso al sistema de bases de datos mediante una interfaz específica de ese lenguaje.

27.3.3.5. Interfaz de programación para servidores

La interfaz de programación para servidores (*server programming interface*: SPI) es una interfaz para programación de aplicaciones que permite que las funciones de C definidas por los usuarios (consulte la Sección 27.3.3.2) ejecuten comandos SQL arbitrarios dentro de las funciones. Esto ofrece a los escritores de funciones definidas por los usuarios la posibilidad de implementar en C solo las partes fundamentales, así como de aprovechar con facilidad toda la potencia del motor de bases de datos relacionales para hacer la mayor parte del trabajo.

27.4. Gestión de transacciones en PostgreSQL

El control de concurrencia de PostgreSQL implementa tanto el aislamiento de instantáneas como el bloqueo de dos fases. El protocolo empleado depende del tipo de instrucción en ejecución. Para las instrucciones LMD[1] se emplea la técnica de aislamiento de instantáneas que se presenta en la Sección 15.7; el aislamiento de instantáneas es el esquema de control de concurrencia multiversión (MVCC) en PostgreSQL. El control de concurrencia para las instrucciones LDD, por su parte, se basa en el bloqueo de dos fases estándar.

27.4.1. Control de concurrencia en PostgreSQL

Dado que los detalles del protocolo MVCC de PostgreSQL dependen del *nivel de aislamiento* solicitado por la aplicación, se comenzará con una visión general de los niveles de aislamiento que ofrece PostgreSQL. Luego se describirán las ideas principales subyacentes al esquema MVCC de PostgreSQL, a lo que seguirá la discusión de su implementación en PostgreSQL y algunas implicaciones del MVCC. Esta sección concluirá con una visión general de los bloqueos para las instrucciones LDD y una discusión del control de concurrencia para los índices.

27.4.1.1. Niveles de aislamiento en PostgreSQL

La norma de SQL define tres niveles de consistencia débiles, además del nivel de consistencia secuenciable, en el que se basa la mayor parte del estudio en este libro. La finalidad de ofrecer los niveles de consistencia débil es permitir un mayor grado de concurrencia para las aplicaciones que no necesiten las sólidas garantías que ofrece la secuencialidad. Entre los ejemplos de estas aplicaciones están las transacciones duraderas que recopilan estadísticas de la base de datos y cuyos resultados no tienen por qué ser muy precisos.

La norma SQL define los diferentes niveles de aislamiento en términos de tres fenómenos que violan la secuencialidad. Esos tres fenómenos se denominan *lectura sucia*, *lectura irrepetible* y *lectura fantasma*, y se definen de la manera siguiente:

- **Lectura sucia.** La transacción lee valores escritos por otra transacción que todavía no se ha comprometido.
- **Lectura irrepetible.** Una transacción lee dos veces el mismo objeto durante la ejecución y encuentra un valor diferente en la segunda lectura, aunque la transacción no haya cambiado ese valor mientras tanto.
- **Lectura fantasma.** Una transacción vuelve a ejecutar una consulta que devuelve un conjunto de filas que satisfacen una condición de búsqueda y descubre que el conjunto de filas que satisfacen la condición ha cambiado como consecuencia de otra transacción comprometida recientemente (se puede hallar una explicación más detallada de este fenómeno en la Sección 15.8.3).

Debería resultar evidente que cada uno de los fenómenos anteriores viola el aislamiento de las transacciones y, por tanto, violan la secuencialidad. La Figura 27.5 muestra la definición de los cuatro niveles de aislamiento de SQL especificados en la norma SQL: lectura no comprometida, lectura comprometida, lectura repetible y secuenciable; en relación con estos fenómenos. PostgreSQL soporta dos de los cuatro niveles de aislamiento, la lectura comprometida, que es el nivel de aislamiento predeterminado en PostgreSQL, y secuenciable. Sin embargo, la implementación de PostgreSQL del nivel de aislamiento secuenciable usa el aislamiento de instantánea, que no es realmente secuenciable, como ya se trató anteriormente en la Sección 15.7.

Nivel de aislamiento	*Lectura sucia*	*Lectura irrepetible*	*Lectura fantasma*
Lectura no comprometida	Posible	Posible	Posible
Lectura comprometida	No	Posible	Posible
Lectura repetida	No	No	Posible
Secuenciable	No	No	No

Figura 27.5. Definición de los cuatro niveles de aislamiento de la norma SQL.

27.4.1.2. Control de concurrencia para los comandos LMD

El esquema MVCC que usa PostgreSQL es una implementación del protocolo de aislamiento de instantáneas que se trató en la Sección 15.7. La idea principal que subyace a MVCC es conservar diferentes versiones de cada fila, que corresponden a diversas instancias de esa fila en diferentes momentos. Ello permite que una transacción vea una **instantánea** consistente de los datos, seleccionando la versión más reciente de cada fila comprometida antes de realizar la instantánea. El protocolo MVCC utiliza instantáneas para asegurar que cada transacción solo vea las versiones de los datos que sean consistentes con la vista de la base de datos: antes de ejecutar un comando la transacción selecciona una instantánea de los datos y procesa las versiones de fila que se encuentran en la instantánea o se crearon por comandos anteriores de esa misma transacción. Esta vista de los datos es «coherente», ya que solo considera transacciones completas, pero la instantánea no necesariamente es igual al estado real de los datos.

La razón de emplear MVCC es que los lectores nunca bloquean a los escritores, y viceversa. Los lectores tienen acceso a la versión más reciente de las filas que forman parte de la instantánea de la transacción. Los escritores crean su propia copia independiente de la fila que se va a actualizar. La Sección 27.4.1.3 muestra que el único conflicto que hace que se bloquee una transacción surge si dos escritores intentan actualizar la misma fila. Por el contrario, con el enfoque estándar de bloqueo de dos fases, puede que se bloqueen tanto los lectores como los escritores, ya que solo hay una versión de cada objeto de datos y tanto las operaciones de lectura como las de escritura deben obtener un bloqueo antes de tener acceso a los datos.

El esquema MVCC de PostgreSQL implementa la versión primero-actualizar-gana del protocolo de aislamiento de instantánea, adquiriendo bloqueos exclusivos sobre las filas que se escriben, pero usando una instantánea (sin ningún bloqueo) cuando se leen filas; se realiza una validación adicional cuando se obtienen bloqueos exclusivos, como se describió anteriormente en la Sección 15.7.

27.4.1.3. Implementación de MVCC en PostgreSQL

En el núcleo MVCC de PostgreSQL se halla el concepto de *visibilidad de tuplas*. Las tuplas de PostgreSQL hacen referencia a versiones de las filas. La visibilidad de las tuplas define qué versión de las potencialmente numerosas versiones de cada fila de una tabla es válida en el contexto de una sentencia o transacción dada. Una transacción determina la visibilidad de una tupla de acuerdo con la instantánea de la base de datos que se seleccione antes de ejecutar un comando.

1 Las instrucciones LMD son instrucciones que actualizan o leen los datos de una tabla, es decir, **select, insert, update, fetch** y **copy.** Las instrucciones LDD afectan a toda la tabla; por ejemplo, pueden eliminarla o cambiar su esquema. Las instrucciones LDD y algunas otras instrucciones exclusivas de PostgreSQL se estudiarán más adelante en esta sección.

Una tupla es visible para una transacción T_A si se cumplen las dos condiciones siguientes:

1. La tupla fue creada por una transacción que se comprometió antes de que la transacción *T* tomara su instantánea.
2. Las actualizaciones de la tupla (si es que se ha llevado a cabo alguna) se ejecutaron por una transacción en cualquiera de las siguientes situaciones:
   - Se abortó.
   - Comenzó a ejecutarse después de que *T* tomara su instantánea.
   - Estaba activa cuando *T* tomó su instantánea.

Para ser preciso, una tupla es visible para *T* si fue creada por *T*, y *T* no la ha actualizado posteriormente. Se omiten los detalles de este caso especial para simplificar.

El objetivo de las condiciones anteriores es asegurarse de que cada transacción solo tenga acceso a los datos que se comprometieron en el momento en que la transacción comenzó a ejecutarse. PostgreSQL mantiene las siguientes estructuras de datos para comprobar eficientemente estas condiciones:

- Se asigna un *identificador de transacción*, que sirve simultáneamente como marca de tiempo, en el momento del comienzo de la transacción. PostgreSQL utiliza un contador lógico (como se describe en la Sección 15.4.1) para la asignación de los identificadores de transacciones.
- Un archivo de registro denominado *pg_clog* contiene el estado actual de cada transacción. El estado puede ser en progreso, comprometida o abortada.
- Cada tupla de una tabla tiene una cabecera con tres campos: *xmin*, que contiene el identificador de la transacción que ha creado la tupla y que, por tanto, se denomina también *identificador de transacción creadora*; *xmax*, que contiene el identificador de transacción de la transacción de sustitución o eliminación (o *null*, si no se elimina o sustituye) y que también se conoce como *identificador de transacción expiradora*, y un enlace hacia adelante a nuevas versiones de la misma fila lógica, si es que hay alguna.
- Se crea una estructura de datos *SnapshotData* bien en el momento inicial de la transacción, bien en el momento inicial de la consulta, según el nivel de aislamiento (que se describe con más detalle más adelante). El objetivo principal es decidir si una tupla es visible para el comando actual. La estructura de datos *SnapshotData* contiene información sobre el estado de las transacciones cuando se crean, que incluye una lista de las transacciones activas y *xmax*, un valor igual a 1 + el *ID* más alto de cualquier transacción que haya comenzado hasta ese momento. El valor de *xmax* sirve como «corte» de las transacciones que se pueden considerar visibles.

La Figura 27.6 muestra estas estructuras de datos mediante un ejemplo sencillo que implica a una base de datos con una sola tabla, la tabla *departamento* de la Figura 27.7. La tabla *departamento* contiene tres columnas, el nombre del departamento, el edificio donde está ubicado y su presupuesto. La Figura 27.6 muestra un fragmento de la tabla *departamento* que solo contiene (las versiones de) la fila del departamento de Física. Las cabeceras de las tuplas indican que la fila la creó originalmente la transacción 100, y que posteriormente la actualizaron las transacciones 102 y 106. La Figura 27.6 muestra también un fragmento del correspondiente archivo de registro *pg_log*. Según el archivo *pg_log*, las transacciones 100 y 102 están comprometidas, mientras que las transacciones 104 y 106 se hallan en progreso.

Dada la información de estado anterior, las dos condiciones que hay que satisfacer para que una tupla resulte visible pueden reescribirse de la siguiente manera:

1. El identificador de transacción creadora de la cabecera de la tupla:
   a. es una transacción comprometida de acuerdo con el archivo *pg_log*, *y*
   b. es menor que el contador de transacciones *xmax* almacenado al comienzo de la consulta en *SnapshotData*, *y*
   c. no estaba procesándose al comienzo de la consulta según *SnapshotData*.
2. El identificador de transacción expiradora, si existe, puede ser:
   a. una transacción abortada según el archivo *pg_clog*, *o*
   b. mayor o igual que el contador de transacciones *xmax* almacenado por *SnapshotData*, *o*
   c. una de las transacciones activas guardadas en *SnapshotData*.

Considere la base de datos de ejemplo de la Figura 27.6 y suponga que el *SnapshotData* que usa la transacción 104 simplemente usa 103 como identificador contador de transacciones *xmax* y no muestra que hay transacciones anteriores activas. En este caso, la única versión de la fila correspondiente al departamento de Física que es visible para la transacción 104 es la segunda versión de la tabla, creada por la transacción 102. La primera versión, creada por la transacción 100, no es visible, dado que viola la condición 2: el identificador de transacción expiradora de esta tupla es 102, que corresponde a una transacción que no está abortada y que tiene un identificador de transacción menor o igual a 103. La tercera versión de la tupla Física no es visible, dado que la creó la transacción 106, que tiene un identificador de transacción mayor que la transacción 104, lo que implica que esta versión no se había comprometido en el momento en que se creó *SnapshotData*. Además, la transacción 106 sigue procesándose, lo que viola otra de las condiciones. La segunda versión de la fila cumple todas las condiciones de visibilidad de las tuplas.

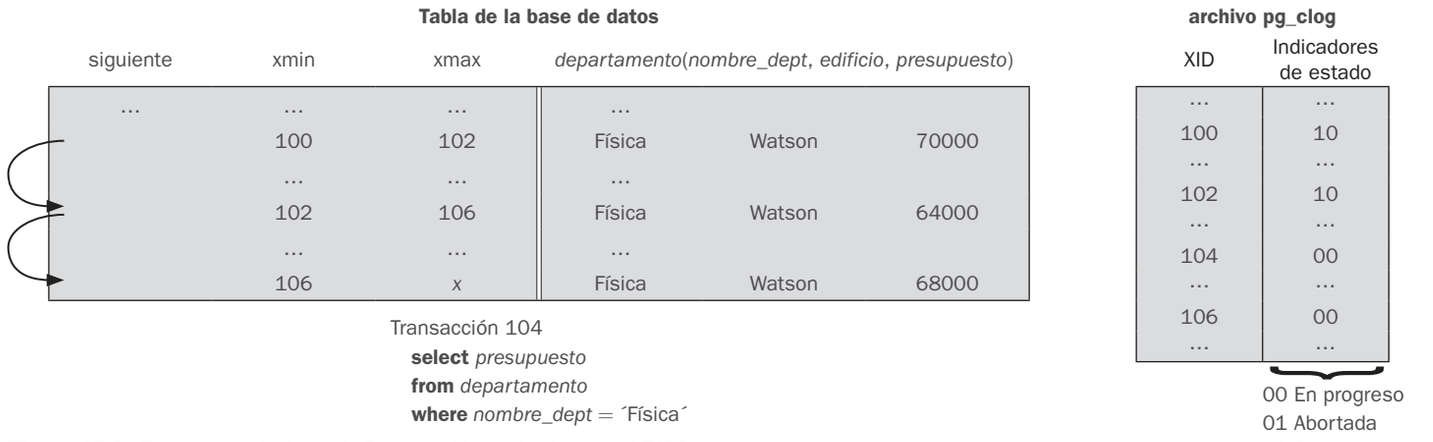


Figura 27.6. Estructuras de datos de PostgreSQL empleadas para MVCC.

nombre_dept	*edificio*	*presupuesto*
Biología	Watson	90000
Informática	Taylor	100000
Electrónica	Taylor	85000
Finanzas	Painter	120000
Historia	Painter	50000
Música	Packard	80000
Física	Watson	70000

Figura 27.7. La relación *departamento*.

Los detalles del modo en que el MVCC de PostgreSQL interactúa con la ejecución de las sentencias de SQL dependen de si la sentencia es **insert, select, update** o **delete.** El caso más sencillo es el de la sentencia **insert,** que simplemente crea una nueva tupla según los datos de la sentencia, inicializa la cabecera de la tupla (el ID de creación) e inserta la nueva tupla en la tabla. A diferencia del caso de los bloqueos de dos fases, no hay esencialmente ninguna interacción con el protocolo de control de concurrencia durante la ejecución de la sentencia, salvo que necesiten comprobar las condiciones de integridad, como que sea única, o las restricciones de clave externa.

Cuando el sistema ejecuta una sentencia **select, update** o **delete,** la interacción con el protocolo MVCC depende del nivel de aislamiento especificado por la aplicación. Si el nivel de aislamiento es lectura comprometida, el procesamiento de una nueva sentencia comienza con la creación de una nueva estructura de datos *SnapshotData* (independientemente de si la sentencia comienza una nueva transacción o forma parte de otra ya existente). A continuación, el sistema identifica las *tuplas objetivo*; es decir, las tuplas que son visibles con respecto a *SnapshotData* y cumplen los criterios de búsqueda de la sentencia. En el caso de las sentencias **select,** el conjunto de tuplas objetivo constituye el resultado de la consulta.

En el caso de las sentencias **update** o **delete,** en el modo de lectura comprometida se necesita un paso adicional tras identificar las tuplas objetivo, antes de que pueda tener lugar la verdadera operación de actualización o de eliminación. El motivo es que la visibilidad de las tuplas solo garantiza que la tupla ha sido creada por una transacción comprometida antes del inicio de la instrucción **update/delete.** No obstante, es posible que, desde el inicio de la consulta, la tupla haya sido actualizada o eliminada por otra transacción que se esté ejecutando de manera concurrente. Esto puede detectarse examinando el identificador de transacción expiradora de la tupla. Si el identificador de transacción expiradora corresponde a una transacción que todavía esté ejecutándose, es necesario esperar a que se complete esa transacción. Si la transacción se aborta, la sentencia **update** o **delete** puede seguir adelante y llevar a cabo la verdadera operación de actualización o de eliminación. Si la transacción se compromete, el criterio de búsqueda de la sentencia debe volver a evaluarse y, solo si la tupla sigue cumpliendo los criterios, puede llevarse a cabo la actualización o la eliminación. Si la fila se va a eliminar, el paso principal es actualizar el ID de transacción expiradora de la tupla antigua. Una actualización de una fila también realiza este paso y, adicionalmente, crea una nueva versión de la fila, establece el ID de transacción creadora y pone el enlace hacia delante de la tupla antigua a la referencia de la nueva.

Volviendo al ejemplo de la Figura 27.6, la transacción 104, que solo consiste en una sentencia **select,** identifica la segunda versión de la fila Física como tupla objetivo y la devuelve de manera inmediata. Si la transacción 104 fuera una sentencia de actualización, por ejemplo, que intentara incrementar el presupuesto del departamento de Física en algún importe, tendría que esperar a que la transacción 106 se completara. Luego volvería a evaluar la condición de búsqueda y, solo si siguiera cumpliéndose, seguiría adelante con la actualización.

Emplear el protocolo descrito anteriormente para las instrucciones **update** y **delete** solo proporciona el nivel de aislamiento de lectura comprometida. La secuencialidad puede violarse de varias maneras. En primer lugar, son posibles lecturas no repetibles. Dado que cada consulta de una transacción puede ver distintas instantáneas de la base de datos, una consulta de la transacción podría ver el efecto de un comando **update** completado mientras no eran visibles consultas anteriores de la misma transacción. Siguiendo la misma línea de pensamiento, son posibles las lecturas fantasmas cuando una relación se modifica entre consultas.

Con objeto de proporcionar el nivel de aislamiento secuenciable de PostgreSQL, MVCC de PostgreSQL elimina las violaciones de la secuencialidad de dos maneras: en primer lugar, cuando determina la visibilidad de las tuplas, todas las consultas de una misma transacción utilizan una instantánea del comienzo de la transacción, en lugar de usar la del comienzo de cada consulta. Así, las sucesivas consultas de una misma transacción siempre ven los mismos datos.

En segundo lugar, el modo en que se procesan las actualizaciones y las eliminaciones es diferente en el modo secuenciable y en el modo de lectura comprometida. Al igual que en el modo de lectura comprometida, las transacciones esperan tras identificar una fila objetivo visible que cumple la condición de búsqueda y está siendo actualizada o eliminada por otra transacción concurrente. Si la transacción concurrente que está ejecutando la actualización o la eliminación se aborta, la transacción que se halla en espera puede seguir adelante con su propia actualización. No obstante, si la transacción concurrente se compromete, no hay manera de que PostgreSQL garantice la secuencialidad de la transacción en espera. Por tanto, la transacción en espera retrocede y devuelve el mensaje de error «*can´t serialize access due to concurrent update*» (no se puede secuenciar el acceso debido a una actualización concurrente).

El manejo correcto de los mensajes de error como el anterior depende de cada aplicación, abortando la transacción correspondiente y reiniciando toda la transacción desde el principio. Observe que los retrocesos debidos a problemas de secuencialidad solo son posibles para las sentencias **update** y **delete.** Sigue cumpliéndose que las sentencias **select** no entran nunca en conflicto con otras transacciones.

27.4.1.4. Repercusiones del empleo de MVCC

El empleo del esquema MVCC de PostgreSQL tiene repercusiones en tres áreas diferentes: (1) Se le asigna carga extra al gestor de almacenamiento, ya que necesita mantener diferentes versiones de las tuplas; (2) hace falta extremar el cuidado al desarrollar aplicaciones concurrentes, ya que el MVCC de PostgreSQL puede generar sutiles, pero importantes, diferencias en el modo en que se comportan las transacciones concurrentes respecto a los sistemas en que se utiliza el bloqueo de dos fases estándar; (3) el rendimiento de PostgreSQL depende de las características de la carga de trabajo que se ejecute en él. Las repercusiones del MVCC de PostgreSQL se describen a continuación con más detalle.

La creación y almacenamiento de varias versiones de cada fila puede llevar a un consumo excesivo de almacenamiento. Para aliviar este problema, PostgreSQL libera periódicamente espacio mediante la identificación y eliminación de versiones que no pueden ser visibles a ninguna transacción activa ni futura y, por tanto, ya no resultan necesarias. La tarea de liberar espacio no es trivial, ya que los índices pueden hacer referencia a la ubicación de tuplas que no son necesarias, por lo que dichas referencias deben ser eliminadas antes de reutilizar el espacio. Para eliminar este problema, PostgreSQL evita indexar múltiples versiones de una tupla con los mismos atributos de índice. Esto permite que el espacio que ocupan tuplas no indexadas se libere de forma eficiente por cualquier transacción que las encuentre.

Para una reutilización más agresiva del espacio, PostgreSQL proporciona el comando **vacuum,** que actualiza correctamente los índices de cada tupla que se libera. El comando **vacuum** se ejecuta en segundo plano, pero también pueden invocarlo directamente los usuarios. El comando **vacuum** ofrece dos modos diferentes de operación: el comando **vacuum** sencillo, que identifica las tuplas que no son necesarias y pone el espacio disponible para reutilizarlo. Esta forma del comando puede operar en paralelo con la lectura y la escritura normales de las tablas. El comando **vacuum full** lleva a cabo un procesamiento más amplio, que incluye el traslado de tuplas de unos bloques a otros para intentar compactar las tablas en un número mínimo de bloques de disco. Esta modalidad es mucho más lenta y exige el bloqueo exclusivo de cada tabla mientras se procesa.

Debido al empleo del control de concurrencia multiversión en PostgreSQL, el traslado de aplicaciones desde otros entornos a PostgreSQL puede exigir algunas precauciones adicionales para garantizar la consistencia de los datos. Por ejemplo, considere una transacción T_A que ejecuta una sentencia **select.** Dado que los lectores de PostgreSQL no bloquean los datos, los datos leídos y seleccionados por T_A pueden ser sobrescritos por otra transacción concurrente, T_B, mientras que T_A todavía se está ejecutando. En consecuencia, puede que parte de los datos que T_A devuelve ya no estén actualizados en el momento de completarse T_A. Puede que T_A devuelva filas que otras transacciones hayan modificado o eliminado mientras tanto. Para garantizar la validez de las filas y protegerlas de las transacciones concurrentes, las aplicaciones deben utilizar **select for share** o adquirir de manera explícita un bloqueo con el comando **lock table** correspondiente.

El enfoque que PostgreSQL da al control de concurrencia proporciona un rendimiento óptimo para las cargas de trabajo que contienen muchas más lecturas que actualizaciones ya que, en ese caso, las posibilidades de que dos actualizaciones entren en conflicto y den lugar al retroceso de una transacción son muy bajas. El bloqueo de dos fases puede ser más eficiente para algunas cargas de trabajo intensivas en actualizaciones, pero esto depende de muchos factores, como el tamaño de las transacciones y la frecuencia de los bloqueos.

27.4.1.5. Control de la concurrencia LDD

Los mecanismos de MVCC descritos en la sección anterior no protegen a las transacciones de las operaciones que afectan a tablas completas como, por ejemplo, las transacciones que descartan una tabla o modifican su esquema. Para ello, PostgreSQL ofrece bloqueos explícitos que los comandos LDD se ven obligados a adquirir antes de comenzar su ejecución. Estos bloqueos siempre se basan en las tablas (en vez de basarse en las filas) y se adquieren y liberan de acuerdo con el estricto protocolo de bloqueo de dos fases.

La Figura 27.8 muestra una lista de todos los tipos de bloqueos de PostgreSQL, los bloqueos con los que entran en conflicto y los comandos que los utilizan (el comando **create index concurrently** se trata en la Sección 27.5.2.3). Los nombres de los tipos de bloqueo suelen ser históricos y no reflejan necesariamente el uso que se les da. Por ejemplo, todos los bloqueos son bloqueos del nivel de tabla, aunque algunos tengan en el nombre la palabra «fila». Los comandos LMD solo adquieren bloqueos de los tipos 1, 2 y 3. Estos tres tipos de bloqueo son compatibles entre sí, ya que MVCC se preocupa de proteger esas operaciones unas de otras. Los comandos LMD solo adquieren esos bloqueos para protegerse de comandos LDD.

Aunque el principal objetivo es proporcionar control interno de la concurrencia a PostgreSQL para los comandos LDD, las aplicaciones de PostgreSQL también pueden adquirir de manera explícita todos los bloqueos de la Figura 27.8 mediante el comando **lock table.**

Nombre del bloqueo	*En conflicto con*	*Adquirido por*
ACCESS SHARE	ACCESS EXCLUSIVE	consulta **select**
ROW SHARE	EXCLUSIVE ACCESS EXCLUSIVE	consulta **select for update** consulta **select for share**
ROW EXCLUSIVE	SHARE SHARE ROW EXCLUSIVE EXCLUSIVE ACCESS EXCLUSIVE	**update** **delete** consultas **insert**
SHARE UPDATE EXCLUSIVE	SHARE UPDATE EXCLUSIVE SHARE SHARE ROW EXCLUSIVE EXCLUSIVE ACCESS EXCLUSIVE	**vacuum** **analyze** **create index concurrently**
SHARE	ROW EXCLUSIVE SHARE UPDATE EXCLUSIVE SHARE ROW EXCLUSIVE EXCLUSIVE ACCESS EXCLUSIVE	**create index**
SHARE ROW EXCLUSIVE	ROW EXCLUSIVE SHARE UPDATE EXCLUSIVE SHARE SHARE ROW EXCLUSIVE EXCLUSIVE ACCESS EXCLUSIVE	—
EXCLUSIVE	Todos excepto ACCESS SHARE	—
ACCESS EXCLUSIVE	Todos los modos	**drop table** **alter table** **vacuum full**

Figura 27.8. Modos de bloqueo en el nivel de tabla.

Los bloqueos se registran en una tabla de bloqueos que se implementa como una tabla asociativa en memoria compartida que utiliza como claves el tipo y el identificador del objeto que se bloquea. Si una transacción desea adquirir un bloqueo sobre un objeto que otra transacción retiene en un modo en conflicto, tiene que esperar hasta que se libere el bloqueo. Las esperas de bloqueo se implementan mediante semáforos, cada uno asociado con una única transacción. Cuando existe un tiempo de espera debido a un bloqueo, realmente es consecuencia del semáforo asociado con la transacción que tiene el bloqueo. Una vez el titular del bloqueo lo libera, se lo comunica a la transacción que está esperando mediante el semáforo. Al implementar las esperas por bloqueo en base al poseedor de cada bloqueo, en vez de hacerlo por objeto bloqueado, PostgreSQL necesita al menos un semáforo por cada transacción concurrente, en lugar de uno por cada objeto bloqueable.

La detección de interbloqueos en PostgreSQL se basa en plazos. De manera predeterminada, se activa la detección de interbloqueos si una transacción ha estado esperando un bloqueo durante más de un segundo. El algoritmo de detección de interbloqueos crea un grafo de esperas en términos de la información de la tabla de bloqueos y busca en ese grafo dependencias circulares. Si encuentra alguna, lo que significa que se ha detectado un interbloqueo, la transacción que desencadenó el interbloqueo se aborta y devuelve un error al usuario. Si no se detecta ningún ciclo, la transacción sigue esperando en el bloqueo. A diferencia de algunos sistemas comerciales, PostgreSQL no ajusta el parámetro de plazo de bloqueo dinámicamente, pero permite que el administrador lo ajuste manualmente. Lo ideal sería escoger este parámetro del orden del tiempo de vida de las transacciones, con objeto de optimizar el equilibrio entre el tiempo que se tarda en detectar un interbloqueo y el trabajo perdido al ejecutar el algoritmo de detección de interbloqueos cuando no se ha producido ninguno.

27.4.1.6. Bloqueos e índices

Todos los tipos actuales de índices de PostgreSQL permiten el acceso concurrente por múltiples transacciones. Normalmente se permite con bloqueos de nivel de página, por lo que diferentes transacciones pueden acceder al índice en paralelo si no solicitan bloqueos en conflicto sobre una página. Estos bloqueos se mantienen, normalmente, durante un breve periodo para evitar el interbloqueo, con la excepción de los índices asociativos, que bloquean las páginas durante periodos mayores y pueden participar en los interbloqueos.

27.4.2. Recuperación

Históricamente, PostgreSQL no usaba un registro histórico de escritura anticipada (*write-ahead logging*: WAL) para la recuperación y, por tanto, no podía garantizar la consistencia en caso de fallo. Los fallos del sistema pueden acabar dando lugar a estructuras de índices inconsistentes o, peor aún, a tablas con sus contenidos completamente corruptos, debido a las páginas de datos escritas parcialmente. En consecuencia, a partir de la versión 7.1, PostgreSQL emplea la recuperación estándar basada en el registro histórico de escritura anticipada. Este enfoque es parecido al de ARIES (Sección 16.8), pero la recuperación en PostgreSQL se simplifica en algunos aspectos debido al protocolo MVCC.

En primer lugar, en PostgreSQL la recuperación no necesita deshacer los efectos de las transacciones abortadas. Las transacciones en proceso de aborto realizan una entrada en el archivo *pg_clog* que registra el hecho de que está en proceso de aborto. En consecuencia, todas las versiones de las filas que deja atrás no serán nunca visibles para las demás transacciones. El único caso en que este enfoque puede acabar dando lugar a problemas se produce cuando una transacción aborta debido a un fallo del proceso PostgreSQL correspondiente y ese proceso no tiene la posibilidad de crear la entrada *pg_clog* antes del fallo del sistema. PostgreSQL maneja esto de la siguiente manera: siempre que una transacción comprueba el estado de otra en el archivo *pg_clog* y descubre que el estado es «en progreso», comprueba si la transacción se está ejecutando realmente en alguno de los procesos de PostgreSQL. Si ningún proceso de PostgreSQL esta ejecutando la transacción, se deduce que el proceso correspondiente ha fallado y la entrada de *pg_clog* de la transacción se actualiza como «abortada».

En segundo lugar, la recuperación basada en el registro histórico de escritura anticipada se simplifica debido a que MVCC de PostgreSQL ya realiza un seguimiento de parte de la información necesaria para el registro WAL. Más concretamente, no hace falta registrar el inicio, el compromiso y el aborto de las transacciones, ya que MVCC registra el estado de cada transacción en *pg_clog*.

27.5. Almacenamiento e índices

El enfoque de PostgreSQL, en cuanto a la disposición y el almacenamiento de los datos, tiene dos objetivos (1) una implementación sencilla y limpia y (2) la facilidad de administración. Con objeto de conseguir estos objetivos, PostgreSQL solo soporta sistemas de archivos «cocinados», lo que excluye el empleo del emplazamiento físico de los datos en particiones de disco. PostgreSQL mantiene una lista de directorios en la jerarquía de archivos para utilizarlos en el almacenamiento, lo que suele denominarse **espacio de tabla.** Las instalaciones de PostgreSQL se inicializan con una tabla de espacio por defecto y puede que a veces sea necesario aumentarlo. Cuando se crea una tabla, un índice o una base de datos completa, el usuario puede especificar la tabla de espacio en que se guardarán los archivos. Resulta especialmente útil para crear varias tablas de espacio si residen en distintos dispositivos físicos, por lo que los dispositivos más rápidos se pueden dedicar a los datos que tienen una mayor demanda. Más aún, a los datos que se almacenan en discos diferentes se puede acceder en paralelo de un modo más eficiente.

El diseño del sistema de almacenamiento de PostgreSQL puede crear algunas limitaciones de rendimiento, debido a la relación entre PostgreSQL y el sistema de archivos. El empleo de sistemas de archivos cocinados da lugar a la utilización de memorias intermedias dobles, en las que cada bloque se captura primero del disco a la memoria intermedia del sistema de archivos (en el espacio del núcleo) antes de copiarlo al grupo de memorias intermedias de PostgreSQL. El rendimiento se puede ver limitado porque PostgreSQL guarda los datos en bloques de 8KB, lo que puede que no coincida con el tamaño de bloque del núcleo. Se puede cambiar el tamaño de bloque de PostgreSQL cuando se instala el servidor, pero puede ocasionar consecuencias no deseadas: un bloque pequeño limita la capacidad de PostgreSQL de guardar tuplas grandes de manera eficiente, mientras que los bloques grandes son un desperdicio cuando se accede a una pequeña región de un archivo.

Por otro lado, las empresas modernas utilizan cada vez más sistemas de almacenamiento externos, como los medios de almacenamiento conectados en red y las redes de áreas de almacenamiento, en lugar de discos conectados a los servidores. La filosofía, en este caso, es que el almacenamiento es un servicio cuyo rendimiento se administra y ajusta con más facilidad de manera independiente. El empleo de RAID para conseguir tanto paralelismo como almacenamiento redundante se explica en la Sección 10.3. PostgreSQL puede aprovechar estas tecnologías debido a su confianza en los sistemas de archivos «cocinados». Por tanto, la sensación de muchos desarrolladores de PostgreSQL es que, para una amplia mayoría de aplicaciones y, en realidad, para el público de PostgreSQL, las limitaciones de rendimiento son mínimas y se justifican por su facilidad de administración y de gestión, así como por la simplicidad de su implementación.

27.5.1. Tablas

La unidad principal de almacenamiento de PostgreSQL es la tabla. En PostgreSQL, las tablas se almacenan en «*archivos en montículo*». Estos archivos utilizan una variante del formato estándar de «*página con ranuras*» que se describe en la Sección 10.5. El formato PostgreSQL se muestra en la Figura 27.9. En cada página, a la cabecera le sigue un *array* de «punteros de línea». Cada puntero de línea guarda el desplazamiento (en relación con el comienzo de la página) y la longitud de una tupla concreta de la página. Las tuplas reales se almacenan en orden inverso a los punteros de línea desde el final de la página.

Cada registro de un archivo en montículo queda identificado por un **identificador de tupla** (TID). El TID consta de un identificador de bloque de 4 bytes que especifica la página del archivo que contiene la tupla y un ID de ranura de 2 bytes. El ID de ranura es un índice del *array* de punteros de línea que, a su vez, se utiliza para tener acceso a la tupla.

Aunque esta infraestructura permite que se eliminen o actualicen las tuplas de una página, en el enfoque de MVCC de PostgreSQL ninguna de esas operaciones elimina o sustituye realmente las versiones antiguas de las filas de manera inmediata. Como se ha descrito en la Sección 27.4.1.4, las tuplas expiradas se pueden eliminar físicamente mediante comandos posteriores, lo que genera huecos en la página. La indirección de acceso a las tuplas mediante los punteros de línea permite reutilizar esos huecos.

La longitud de cada tupla habitualmente viene limitada por la de las páginas de datos. Esto hace difícil almacenar tuplas muy largas. Cuando PostgreSQL encuentra una tupla tan grande, intenta **comprimir** (*toast*) los atributos de gran tamaño. En algunos casos la compresión de un atributo se puede acompañar de la compresión del valor. Si no se comprime lo suficiente como para que quepa en una página (lo que suele ser el caso), los datos de las columnas **comprimidas** son sustituidos por un puntero que localiza una versión comprimida de los datos que se almacenan fuera de la página.

27.5.2. Índices

Un índice de PostgreSQL es una estructura de datos que proporciona una asociación dinámica de un predicado de búsqueda en secuencias de TID desde una tabla dada. Las tuplas resultado casan con el predicado de búsqueda, aunque en algunos casos se deba recalcular el predicado en el archivo en montículo. PostgreSQL soporta diferentes tipos de índices, incluyendo los basados en métodos de acceso extensible de usuario. Aunque los métodos de acceso pueden usar un formato de página diferente, todos los índices disponibles en PostgreSQL usan el mismo formato de página con ranuras descrito en la Sección 27.5.1.

27.5.2.1. Tipos de índices

PostgreSQL soporta los siguientes tipos de índices:

- **De árbol B.** El tipo de índice predeterminado es un método de árbol B^+ que es una implementación de los árboles B^+ de alta concurrencia de Lehman-Yao (esto se explica con detalle en la Sección 15.10). Estos índices resultan útiles para las consultas de igualdad y para las de rango sobre datos ordenables, así como para algunas operaciones de coincidencia de patrones como la expresión **like.**
- **De asociación.** Los índices asociativos de PostgreSQL son una implementación de la asociación lineal (para más información sobre los índices de asociación, véase la Sección 11.6.3). Este tipo de índices solo resulta útil para las operaciones de igualdad sencillas. Se ha demostrado que los índices asociativos de PostgreSQL tienen un rendimiento de búsqueda no mejor que el de los árboles B, pero tienen un tamaño y unos costes de mantenimiento considerablemente superiores. Además, los índices asociativos son los únicos índices de PostgreSQL que no admiten recuperación frente a fallos. Por tanto, casi siempre es preferible utilizar índices de árbol B que índices de asociación.
- **GiST.** PostgreSQL soporta un tipo de índices altamente extensible denominado GiST, o árboles de búsqueda generalizados (Generalized Search Trees). Es un método de acceso con estructura de árbol equilibrado estructurado que facilita a los expertos en dominios que estén versados en un tipo de datos concreto (como los datos de imágenes) el desarrollo de índices que mejoren el rendimiento sin tener que tratar con los detalles internos del sistema de bases de datos. Entre los ejemplos de índices creados empleando GiST se incluyen los árboles B y los árboles R, así como índices menos convencionales, como la indexación de cubos multidimensionales y la indexación de texto completo para las consultas de recuperación de información. Se pueden implementar nuevos métodos de acceso GiST creando una clase de operador, como se describe en la Sección 27.3.3.3. Las clases de operadores para GiST son diferentes que para los árboles, ya que cada clase de operador GiST puede tener un conjunto de estrategias que indican los predicados de búsqueda implementados por el índice. GiST también se apoya en siete funciones de soporte para operaciones, como la comprobación de pertenencia al conjunto y la división de conjuntos de entradas para los desbordamientos de página.

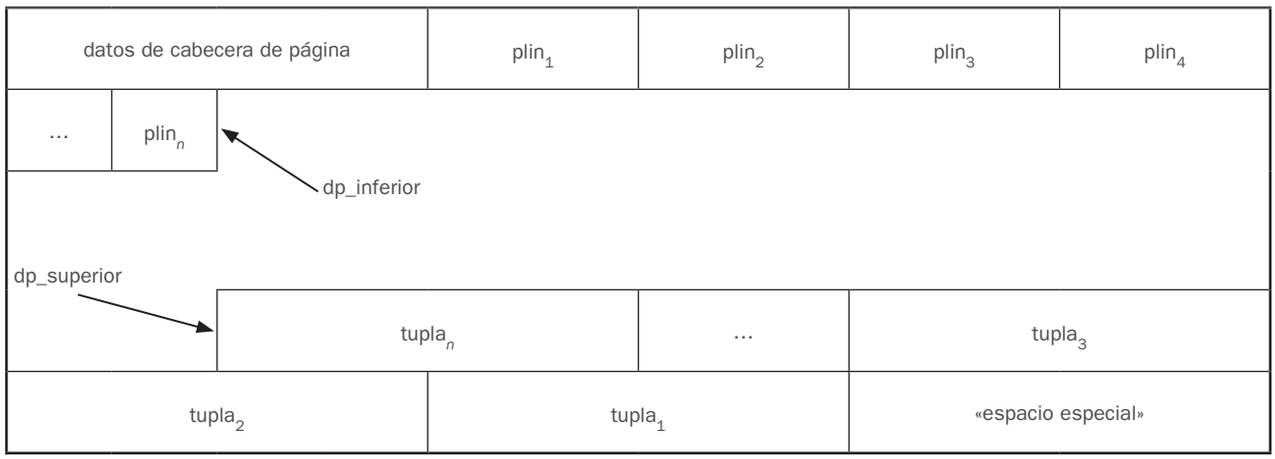


Figura 27.9. Formato de página con ranuras para las tablas de PostgreSQL.

Resulta interesante indicar que la implementación original en PostgreSQL de los árboles R (Sección 25.3.5.3) se sustituyó por clases de operador GiST en la versión 8.2. De esta forma los árboles R tienen la capacidad del registro WAL y capacidades de concurrencia que se añadieron a GiST en la versión 8.1. Como la implementación original de árboles R no contaba con estas capacidades, este cambio muestra las ventajas del método de indexado extensible. Consulte las notas bibliográficas para más información sobre los índices GiST.

- **GIN.** El tipo más nuevo de índice en PostgreSQL es el índice inverso generalizado (*generalized inverted index*: GIN). Un índice GIN interpreta tanto las claves de índices como las claves de búsqueda como conjuntos, lo que adecua el tipo índice con los operadores orientados a conjuntos. Uno de los usos pretendidos para GIN es la indexación de documentos para la búsqueda de texto completo, que se implementa reduciendo los documentos y consultas a conjuntos de términos de búsqueda. Como GiST, un índice GIN se puede extender para manejar cualquier operador de comparación creando una clase de operador con las funciones de soporte apropiadas.

  Para evaluar una búsqueda, GIN identifica eficientemente las claves de índices que se superponen con las claves de búsqueda, y calcula un mapa de bis que indica cuáles de los elementos de la búsqueda son miembros de la clave índice. Esto se lleva a cabo usando funciones de soporte que extrae miembros de un conjunto y compara los miembros individuales. Otra función de soporte decide si se satisface el predicado de búsqueda de acuerdo con el mapa de bits y el predicado original. Si el predicado de búsqueda no se puede resolver sin el atributo indexado completo, la función de decisión debe indicar una coincidencia y volver a comprobar el predicado en el archivo en montículo.

27.5.2.2. Otras variaciones de los índices

Para algunos de los tipos de índices que se han descrito, PostgreSQL soporta más variaciones complejas, como:

- **Índices de varias columnas.** Resultan útiles para conjuntos de predicados que abarcan varias columnas de una tabla. Los índices multicolumna solo se soportan para índices de árbol B y para índices GiST.
- **Índices únicos.** Las restricciones de clave única y de clave principal pueden forzarse mediante el empleo de índices únicos en PostgreSQL. Solo se pueden definir como únicos los índices de árbol B.
- **Índices sobre expresiones.** En PostgreSQL es posible crear índices sobre expresiones escalares arbitrarias de las columnas de las tablas, y no solo sobre columnas concretas. Esto resulta especialmente útil cuando las expresiones en cuestión son «costosas»; es decir, suponen un cálculo complicado definido por los usuarios. Un ejemplo es el soporte de comparaciones que no distinguen entre mayúsculas y minúsculas mediante la definición de un índice sobre la expresión *lower(columna)* y el empleo del predicado *lower(columna)* = «valor» en las consultas. Un inconveniente son los elevados costes de mantenimiento de los índices sobre expresiones.
- **Clases de operadores.** Las funciones de comparación concretas empleadas para crear, mantener y utilizar los índices sobre columnas están ligadas al tipo de datos de cada columna. Cada tipo de datos tiene asociada una «clase de operador» predeterminada (que se describe en la Sección 27.3.3.3) que identifica los operadores reales que se utilizarían normalmente. Aunque este operador predeterminado suele resultar suficiente para la mayor parte de los usos, puede que algunos tipos de datos posean varias clases «significativas». Por ejemplo, al tratar con números complejos, puede que resulte preferible indexar el componente real o el imaginario. PostgreSQL ofrece algunas clases de operador incorporadas para operaciones de comparación de patrones (como **like**) sobre datos de texto que no utilicen las reglas de comparación con locales estándar (es decir, con ordenaciones específicas para determinados idiomas).
- **Índices parciales.** Se trata de índices creados sobre un subconjunto de una tabla definido por un predicado. El índice solo contiene entradas para tuplas que satisfagan el predicado. Los índices parciales resultan adecuados para aquellos casos en que una columna pueda contener gran número de apariciones de un número muy pequeño de valores. En tales casos, no merece la pena indexar los valores comunes, ya que la exploración del índice no es apropiada para búsquedas que requieren un gran subconjunto de la tabla base. Los índices parciales que excluyen los valores frecuentes son pequeños y suponen menos operaciones E/S. Los índices parciales resultan menos costosos de mantener, ya que una fracción importante de las inserciones no participa en ellos.

27.5.2.3. Creación de índices

Se añade un índice a la base de datos usando el comando **create index.** Por ejemplo, la siguiente sentencia de LDD crea un índice de árbol B sobre los sueldos de los profesores:

create index *ind_sueldos* **on** *profesor* (*sueldo*);

Esta sentencia se ejecuta explorando la relación *profesor* y buscando versiones de filas que puedan ser visibles a futuras transacciones, y después ordenando sus atributos de índice y construyendo la estructura índice. Durante este proceso, la transacción de creación mantiene un bloqueo sobre la relación *profesor* que evita la concurrencia de sentencias **insert, delete** o **update.** Cuando el procesamiento termina, el índice está listo para su uso y se libera el bloqueo de la tabla.

El bloqueo que adquiere el comando **create index** puede plantear un cierto inconveniente a algunas aplicaciones en las que es difícil suspender las actualizaciones mientras se construye el índice. Para estos casos, PostgreSQL proporciona la variante **create index concurrently,** que permite la concurrencia de actualizaciones durante la creación del índice. Se realiza mediante un algoritmo de creación más complejo que explora la tabla base dos veces. En la primera exploración de la tabla construye una versión inicial del índice, de forma similar al proceso de construcción normal descrito anteriormente. En este índice pueden faltar tuplas si la tabla se actualiza concurrentemente; sin embargo, el índice está bien formado, por lo que se marca como listo para realizar inserciones. Finalmente, el algoritmo explora la tabla una segunda vez e inserta todas las tuplas que encuentra que aún necesitan indexarse. Esta exploración puede volver a perder de nuevo las tuplas que se actualicen concurrentemente, pero el algoritmo sincroniza con otras transacciones para garantizar que las tuplas que se actualizan durante la segunda exploración se añaden al índice en la transacción de actualización. Por tanto, el índice está listo para su uso tras la segunda exploración de la tabla. Como este enfoque de exploración de dos pasos puede ser costoso, es preferible el comando **create index** si resulta factible suspender las actualizaciones de la tabla temporalmente.

27.6. Procesamiento y optimización de consultas

Cuando PostgreSQL recibe una consulta, esta se analiza primero en una representación interna que sufre una serie de transformaciones que dan lugar a un plan de consulta que el **ejecutor** utiliza para procesar la consulta.

27.6.1. Reescritura de consultas

La primera etapa de la transformación de una consulta es la **reescritura** (*rewrite*) y es la responsable del sistema de **reglas** (*rules*) del sistema de PostgreSQL. Como se explicó en la Sección 27.3, en PostgreSQL los usuarios pueden crear **reglas** que se activan ante diferentes eventos, como instrucciones **update, delete, insert** y **select.** El sistema implementa una vista mediante la conversión de la definición de la vista en una regla **select.** Cuando se recibe una consulta que implica una instrucción **select** sobre la vista, se desencadena la regla **select** para la vista y se reescribe la consulta utilizando la definición de la vista.

Las reglas se registran en el sistema utilizando el comando **create rule,** momento en el cual la información sobre la regla se almacena en el catálogo. Este catálogo se utiliza luego durante la reescritura de la consulta para descubrir todas las reglas candidatas para una consulta dada.

La fase de reescritura trabaja primero con todas las instrucciones **update, delete** e **insert** desencadenando todas las reglas adecuadas. Observe que puede que tales instrucciones sean complejas y contengan cláusulas **select.** En consecuencia, se desencadenan todas las demás reglas que solo contienen instrucciones **select.** Dado que el desencadenamiento de una regla puede hacer que la consulta se reescriba de una manera que exija que se desencadene otra regla, las reglas se comprueban repetidamente en cada forma de la consulta reescrita hasta que se alcanza un punto fijo y no hace falta desencadenar más reglas.

En PostgreSQL no hay reglas predeterminadas, solo las definidas de manera explícita por los usuarios y de forma implícita por las definiciones de las vistas.

27.6.2. Planificación y optimización de consultas

Una vez reescrita la consulta, se somete a la fase de planificación y optimización. En esta fase cada bloque de la consulta se trata de manera aislada y se genera un plan específico para él. Esta planificación comienza de abajo arriba desde la subconsulta más interna de la consulta reescrita, hasta su bloque de consulta más externo.

El optimizador de PostgreSQL está, en su mayor parte, basado en costes. La ideal es generar un plan de acceso cuyo coste estimado sea mínimo. El modelo de costes incluye como parámetros tanto el coste de E/S de la obtención no secuencial de páginas como los costes de CPU del procesamiento de tuplas en montículo, tuplas de índices y predicados sencillos.

El proceso real de optimización se basa en una de las dos formas siguientes:

- **Planificador estándar.** Se trata de un algoritmo de programación dinámica de abajo arriba para la optimización del orden de reunión que se utilizaba en System R, un sistema relacional pionero desarrollado por los centros de investigación de IBM en los años setenta. El algoritmo de programación dinámica del Sistema R se describe en detalle en la Sección 13.4.1. El algoritmo se usa para un bloque de consulta único cada vez.
- **Optimizador genético de consultas.** Cuando el número de tablas en un bloque de consulta es muy grande, el algoritmo de programación dinámica del Sistema R se vuelve muy costoso. A diferencia de otros sistemas comerciales que pasan de manera predeterminada a técnicas ávidas o basadas en reglas, PostgreSQL utiliza un enfoque más radical: un algoritmo genético que inicialmente se desarrolló para resolver problemas en las rutas de viajantes. Hay pruebas anecdóticas del empleo con éxito de la optimización genética de consultas en sistemas en producción para consultas con alrededor de 45 tablas.

Dado que el planificador opera de abajo arriba, puede llevar a cabo ciertas transformaciones en el plan de la consulta a medida que este se genera. Un ejemplo es la frecuente transformación de subconsulta en reunión que aparece en muchos sistemas comerciales (generalmente implementada en la fase de reescritura). Cuando PostgreSQL encuentra una subconsulta no correlacionada (como puede ser la causada por una consulta sobre una vista), suele ser posible «elevar» la subconsulta planificada y mezclarla en el bloque de consulta del nivel superior. No obstante, las transformaciones que empujan las eliminaciones de duplicados hacia bloques de consulta de nivel inferior no suelen ser posibles en PostgreSQL.

La fase de optimización de consultas da lugar a un plan de consultas que es un árbol de operadores relacionales. Cada operador representa una operación concreta sobre uno o varios conjuntos de tuplas. Los operadores pueden ser unarios (por ejemplo, ordenación, agregación), binarios (como la reunión de bucle anidado) o n-arios (por ejemplo, la unión de conjuntos).

En el modelo de costes resulta crucial una estimación precisa del número total de tuplas que se procesarán en cada operador del plan. Esto lo deduce el optimizador en función de las estadísticas que se tienen de cada relación del sistema. Estas estadísticas indican el número total de tuplas para cada relación y aportan información específica sobre cada columna de una relación, como puede ser la cardinalidad de esa columna, una lista de los valores más frecuentes de la tabla y su número de apariciones, así como un histograma que divida los valores de la columna en grupos de igual población (es decir, un histograma de equiprofundidad tal como el que se describe en la Sección 13.3.1). Además, PostgreSQL tiene también una correlación estadística entre las ordenaciones física y lógica en las filas de los valores de las columnas. El ABD debe garantizar que estas estadísticas estén al día mediante la ejecución periódica del comando **analyze.**

27.6.3. Ejecutor de consultas

El módulo ejecutor es responsable del procesamiento de los planes de consultas que produce el optimizador. El ejecutor sigue el modelo **iterador** (*iterator*) con un conjunto de cuatro funciones implementadas para cada operador (open, next, rescan y close). Los iteradores también se estudian como parte de los cauces bajo demanda en la Sección 12.7.2.1. Los iteradores de PostgreSQL tienen una función adicional, rescan, que se utiliza para reiniciar subplanes (por ejemplo, para el bucle interno de una reunión) con parámetros como pueden ser los rangos de las claves del índice.

Algunos de los operadores importantes del ejecutor pueden catalogarse de la siguiente manera:

1. **Métodos de acceso.** Los métodos de acceso que se utilizan en PostgreSQL para recuperar datos de objetos a partir de un disco son las exploraciones secuenciales de archivos en montículo, las exploraciones de índices y las de índices de mapas de bits.
   - **Exploraciones secuenciales.** Las tuplas de cada relación se exploran secuencialmente desde el primer bloque del archivo hasta el último. Cada tupla se devuelve al invocador solo si es «visible» de acuerdo con las reglas de aislamiento de transacciones de la Sección 27.4.1.3.
   - **Exploraciones de índices.** Dada una condición de búsqueda como un predicado sobre un rango o una igualdad, este método de acceso devuelve un conjunto de tuplas coincidentes del archivo en montículo asociado. El operador procesa una tupla cada vez, empezando por la lectura de una entrada desde el índice y obteniendo la tupla correspondiente del archivo en montículo. En el peor de los casos puede hacer que se obtenga una página aleatoria para cada tupla.
   - **Exploración de índices de mapas de bits.** Una exploración de índices de mapas de bits reduce el riesgo de obtener páginas excesivamente aleatorias de la exploración de índices. Se consigue procesando las tuplas en dos fases. En

la primera fase se leen todas las entradas de los índices y se guardan los ID de tuplas en montículo en un mapa de bits; en la segunda fase se obtienen las tuplas en montículo coincidentes en orden secuencial. Esto garantiza que solo se accede una vez a cada página del montículo y aumenta la posibilidad de obtener las páginas secuencialmente. Además se pueden mezclar mapas de bits de varios índices y cruzarlos para evaluar predicados boolenanos complejos antes de acceder al montículo.

2. **Métodos de reunión.** PostgreSQL soporta tres métodos de reunión: reuniones por mezcla ordenada, reuniones de bucle anidado (incluidas las variantes de índice de bucle anidado interior) y una reunión híbrida asociativa (Sección 12.5).
3. **Ordenación.** La ordenación externa se implementa en PostgreSQL mediante algoritmos que se explican en la Sección 12.4. Los datos de entrada se dividen en secuencias ordenadas que se mezclan en una mezcla polifásica. Aunque las secuencias iniciales se formen mediante selección por sustitución, se utiliza un árbol de prioridades en lugar de una estructura de datos para fijar el número de registros que se guardan en la memoria. Esto se debe a que PostgreSQL puede trabajar con tuplas que varían considerablemente en tamaño e intenta garantizar la utilización completa del espacio de memoria configurado para la ordenación.
4. **Agregación.** La agregación agrupada en PostgreSQL puede basarse en la ordenación o en la asociación. Cuando el número estimado de grupos diferentes es muy grande, se utiliza la primera y, en caso contrario, se prefiere el enfoque basado en la asociación.

27.6.4. Disparadores y restricciones

En PostgreSQL (a diferencia de algunos sistemas comerciales) las características de las bases de datos activas, como los disparadores y las restricciones, no se implementan en la fase de reescritura. Por el contrario, se implementan como parte del ejecutor de consultas. Cuando el usuario registra los disparadores y las restricciones, los detalles se asocian con la información del catálogo para cada relación e índice adecuados. El ejecutor procesa las instrucciones **update, delete** e **insert** generando repetidamente modificaciones en las tuplas para la relación. Por cada una de esas modificaciones el ejecutor comprueba de manera explícita los disparadores y las restricciones candidatos, y los desencadena o aplica, antes o después de la modificación, según se requiera.

27.7. Arquitectura del sistema

La arquitectura del sistema PostgreSQL sigue el modelo de proceso por transacción. Los sitios PostgreSQL en funcionamiento son gestionados por un proceso coordinador central, denominado **postmaster.** El proceso postmaster es responsable de inicializar y cerrar el servidor, así como de manejar las solicitudes de conexión de nuevos clientes. El postmaster asigna cada nuevo cliente que se conecta a un proceso servidor en segundo plano que es responsable de ejecutar las consultas en nombre del cliente y de devolver los resultados al cliente. Esta arquitectura se describe en la Figura 27.10.

Las aplicaciones cliente pueden conectarse con el servidor de PostgreSQL y remitir consultas mediante alguna de las muchas interfaces de programación de aplicaciones de la base de datos soportadas por PostgreSQL (libpq, JDBC, ODBC, Perl DBD) que se proporcionan como bibliotecas para la parte cliente. Un ejemplo de aplicación cliente es el programa de línea de comandos psql, que se incluye en la distribución PostgreSQL estándar. El postmaster es responsable de manejar las conexiones iniciales de los clientes. Para ello, escucha constantemente si hay nuevas conexiones en un puerto conocido. Tras llevar a cabo los pasos de inicialización, como la autenticación del usuario, el postmaster genera un nuevo proceso de servidor en segundo plano para manejar el nuevo cliente. Tras esta conexión inicial, el cliente solo interactúa con el proceso servidor en segundo plano, remitiendo consultas y recibiendo resultados de consultas. Esta es la esencia del modelo de proceso por conexión adoptado por PostgreSQL.

El proceso servidor en segundo plano es responsable de ejecutar las consultas remitidas por el cliente mediante la realización de los pasos de ejecución de consultas necesarios, incluidos el análisis, la optimización y la ejecución. Cada proceso de servidor en segundo plano solo puede manejar una única consulta a la vez. Para ejecutar más de una consulta en paralelo, la aplicación debe establecer varias conexiones con el servidor.

En cualquier momento puede haber varios clientes conectados al sistema y, por tanto, varios procesos de servidor en segundo plano pueden estar ejecutándose de manera concurrente. Los procesos de servidor en segundo plano tienen acceso a los datos de la base de datos mediante el grupo de memorias intermedias de la memoria principal, que se ubica en la memoria compartida, por lo que todos los procesos tienen la misma vista de los datos. La memoria compartida también se utiliza para implementar otras formas de sincronización entre los procesos de servidor como, por ejemplo, el bloqueo de elementos de datos.

El empleo de memoria compartida como medio de comunicación exige que el servidor de PostgreSQL se ejecute en una sola máquina; los sitios con un solo servidor no pueden extenderse por varias máquinas sin la asistencia de paquetes de terceros como la herramienta de replicación Slony. Sin embargo, se puede construir un sistema de bases de datos sin compartición con un único ejemplar de PostgreSQL ejecutando en cada nodo; de hecho, se han construido varios sistemas de bases de datos paralelos utilizando exactamente esta arquitectura, como se describe en la Sección 18.8.

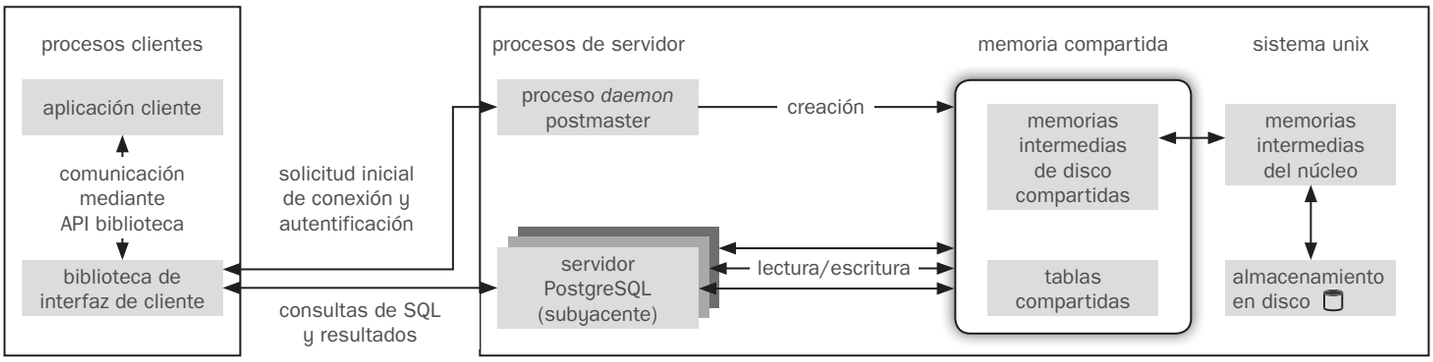


Figura 27.10. Arquitectura del sistema PostgreSQL.

Notas bibliográficas

Se puede encontrar una extensa documentación en línea de PostgreSQL en www.postgresql.org. Este sitio web es la fuente autorizada de información sobre las nuevas versiones de PostgreSQL, que aparecen con frecuencia. Hasta PostgreSQL versión 8, la única manera de ejecutar PostgreSQL bajo Microsoft Windows era utilizar Cygwin. Cygwin es un entorno similar a Linux que permite la generación de aplicaciones Linux a partir de su código para ejecutarlas bajo Windows. Para obtener más detalles consulte www.cygwin.com. Entre los libros sobre PostgreSQL están los de Douglas y Douglas [2003] y Stinson [2002]. Las reglas, tal y como se utilizan en PostgreSQL, se presentaron en Stonebraker et ál. [1990]. La estructura GiST se describe en Hellerstein et ál. [1995].

Muchas herramientas y extensiones para PostgreSQL están documentadas en el sitio web www.pgfoundry.org. Incluyen la biblioteca pgtcl, y la herramienta de administración pgAccess mencionadas al comienzo del capítulo. La herramienta pgAdmin se describe en la web www.pgadmin.org. Las herramientas de diseño de bases de datos TORA y Data Architect se describen en tora.sourceforge.net y www.thekompany.com/products/dataarchitect, respectivamente. Las herramientas de generación de informes GNU Report Generator y GNU Enterprise se describen en www.gnu.org/software/grg y www.gnuenterprise.org, respectivamente. El servidor OLAP de Mondrian se describe en mondrian.pentaho.org.

Una alternativa de código abierto a PostgreSQL es MySQL, que está disponible para usos no comerciales bajo la Licencia Pública General GNU, pero que exige el pago para usos comerciales. Las comparaciones entre las versiones más recientes de los dos sistemas se pueden conseguir fácilmente en la web.

Oracle

28

Hakan Jakobsson

Cuando Oracle se fundó, en 1977, como Software Development Laboratories por Larry Ellison, Bob Miner y Ed Oates, no había productos comerciales de bases de datos relacionales. La compañía, cuyo nombre cambió posteriormente a Oracle, se estableció para construir un sistema de gestión de bases de datos como producto comercial y fue pionera en el mercado de RDBMS, manteniendo el liderazgo en el mercado de bases de datos relacionales desde entonces. Con el paso de los años su producto y los servicios ofrecidos han crecido más allá del servicio de bases de datos relacionales, para incluir aplicaciones y software de soporte.

Además de los productos desarrollados en la compañía, la oferta de Oracle incluye software originalmente desarrollados por compañías que Oracle ha adquirido. Las adquisiciones de Oracle van desde pequeñas compañías a otras muy grandes, con negociación pública, como Peoplesoft, Siebe, Hyperion y BEA. En consecuencia, Oracle dispone de una gran oferta de productos empresariales de software.

Este capítulo se centra en el servidor de bases de datos principal de Oracle y los productos directamente relacionados. Continuamente se desarrollan nuevas versiones de los productos, por lo que las descripciones de estos están sujetas a cambios. El conjunto de características descrito en este capítulo está basado en la primera versión de Oracle11*g*, que es uno de los productos estrella de bases de datos de Oracle.

28.1. Herramientas para el diseño y la consulta de bases de datos

Oracle proporciona una serie de herramientas para el diseño, la consulta, la generación de informes y el análisis de datos para bases de datos, incluyendo OLAP. Estas herramientas, junto con otras herramientas de desarrollo de aplicaciones, forman parte del catálogo de productos software denominado Oracle Fusion Middleware. Entre los productos se incluyen tanto las herramientas tradicionales que usan el lenguaje de programación PL/SQL de Oracle, como otras más nuevas basadas en la tecnología Java/J2EE. Este software soporta los estándares abiertos como SOAP, XML, BPEL y UML.

28.1.1. Herramientas para el diseño de bases de datos

El Oracle *Application Development Framework* (ADF) es un entorno de desarrollo basado en J2EE para el patrón de diseño Modelo-Vista-Controlador. En este entorno de diseño una aplicación consta de varias capas. Las capas del modelo y los servicios de negocio manejan la interacción con las fuentes de datos y contienen la lógica de negocio. La capa de vista maneja la interfaz de usuario y la capa de controlador maneja el flujo de la aplicación así como la interacción entre las otras capas.

La herramienta principal de desarrollo para ADF de Oracle es JDeveloper, que proporciona un entorno integrado de desarrollo con soporte para el desarrollo con Java, XML, PHP, HTML, JavaScript, BPEL, SQL y PL/SQL. Ha incorporado soporte también para modelado con UML.

Oracle Designer es una herramienta de diseño de bases de datos que traslada la lógica de negocio y los flujos de datos en definiciones de esquema y guiones procedimentales para la lógica de la aplicación. Soporta técnicas de modelado como los diagramas E/R, la ingeniería de información y el análisis y diseño orientados a objetos.

Oracle también dispone de la herramienta de desarrollo de aplicaciones para almacenes de datos, Oracle Warehouse Builder. Warehouse Builder es una herramienta para el diseño y desarrollo de todos los aspectos del almacén de datos, incluyendo el diseño de los esquemas, la asociación de datos y las transformaciones, el proceso de la carga de datos y la gestión de los metadatos. Oracle Warehouse Builder soporta tanto 3FN como esquemas en estrella y puede también importar diseños de Oracle Designer. Esta herramienta, junto con las características de la base de datos, como las tablas externas y las funciones de tablas, elimina la necesidad de herramientas de extracción, transformación y carga (ETL) de terceros.

28.1.2. Herramientas de consulta

Oracle proporciona herramientas de consulta, generación de informes y análisis de datos ad hoc, incluyendo OLAP.

Oracle Business Intelligence (BI) Suite es un conjunto de herramientas que comparten una arquitectura común orientada a servicios. Entre sus componentes están un servidor BI, herramientas para la consulta ad hoc y la generación de borradores, informes y alertas. Los componentes comparten infraestructura y servicios para el acceso a los datos y la gestión de los metadatos, y tienen un modelo de seguridad y una herramienta de administración comunes.

El componente para la consulta ad hoc, Oracle BI Answers, es una herramienta interactiva que presenta al usuario una vista lógica de los datos ocultando los detalles de la implementación física. Los objetos disponibles para el usuario se muestran de forma gráfica y el usuario puede crear una consulta con una interfaz de apuntar y hacer clic. Esta consulta lógica se envía al componente Oracle BI Server, que genera la consulta o consultas físicas. Se admiten varias fuentes de datos físicos y la consulta podría combinar datos almacenados en bases de datos relacionales, Olap u hojas de Excel. Los resultados se presentarán en forma de gráficos, informes, tablas pivote y pizarras que se pueden guardar y modificar posteriormente.

28.2. Variaciones y extensiones de SQL

Oracle soporta parcial o totalmente todas las características principales de SQL:2003, con la excepción de las vistas de características y conformidad. Además, Oracle soporta un gran número de construcciones de otros lenguajes, algunos de ellos conformes con las características opcionales de las bases de SQL:2003, mientras que otros son específicos de Oracle en sintaxis o funcionalidad.

28.2.1. Características relacionales orientadas a objetos

Oracle tiene soporte extensivo para constructores relacionales orientados a objetos, incluyendo:

- **Tipos de objetos.** Se soporta un único modelo de herencia para las jerarquías de tipos.
- **Tipos de colecciones.** Oracle soporta **varrays,** que son arrays de longitud variable, y tablas anidadas.
- **Tablas de objetos.** Se utilizan para almacenar objetos mientras se proporciona una vista relacional de los atributos de los objetos.
- **Funciones de tablas.** Son funciones que producen conjuntos de filas como salida y se pueden utilizar en la cláusula **from** de una consulta. Las funciones de tablas se pueden anidar en Oracle. Si una función de tablas se utiliza para expresar algún tipo de transformación de datos, el anidamiento de varias funciones permite que se expresen diversas transformaciones en una única instrucción.
- **Vistas de objetos.** Proporcionan una vista de tablas de objetos virtuales de datos almacenados en una tabla relacional normal. Permite acceder o ver los datos en un estilo orientado a objetos incluso aunque los datos estén realmente almacenados en un formato relacional tradicional.
- **Métodos.** Se pueden escribir en PL/SQL, Java o C.
- **Funciones de agregación definidas por el usuario.** Se pueden utilizar en instrucciones SQL de la misma forma que las funciones incorporadas, tales como **sum** y **count.**

28.2.2. Oracle XML DB

Oracle XML DB proporciona un almacenamiento en la base de datos para datos en XML y soporta muchas funcionalidades de XML, incluyendo XML Schema y XQuery. Se crea a partir del tipo abstracto de datos **XMLType,** que se trata como nativo. XML DB proporciona varias opciones sobre cómo almacenar los datos de este tipo en la base de datos, entre otras:

- Estructurados en formato objeto-relacional. Este formato es normalmente eficiente en espacio y permite el uso de distintas características de bases de datos relacionales estándar, como los índices de árboles B, pero conlleva cierta sobrecarga cuando se asocian documentos en XML a su formato de almacenamiento, y viceversa. Es apropiado, principalmente, para datos en XML que sean altamente estructurados y la asociación incluya un número manejable de tablas y reuniones relacionales.
- No estructurados como un texto. Esta representación no requiere ninguna asociación y proporciona un gran rendimiento cuando se inserta o recupera un documento en XML completo. Sin embargo, no es muy eficiente en espacio y dispone de menor inteligencia de procesado cuando se trabaja sobre partes del documento en XML.
- En almacenamiento binario de XML. Esta representación es un formato binario, tras un análisis, de acuerdo con el XML Schema. Es más eficiente en cuanto al espacio que el almacenamiento no estructurado y puede manejar operaciones contra partes del documento en XML. También es mejor que el formato estructurado en el manejo de los datos que son poco estructurados, pero puede que no sea siempre tan eficiente en espacio. Este formato puede hacer que el procesamiento de sentencias en XQuery sea menos eficiente que si se usa el formato estructurado.

Tanto la representación binaria como la no estructurada se pueden indexar con un tipo especial de índices llamado **XMLIndex.** Este tipo de índice permite indexar fragmentos de documentos según su expresión XPath correspondiente.

Almacenar datos en XML en la base de datos significa que consiguen las ventajas de la funcionalidad de Oracle en áreas como la copia de seguridad, la recuperación, la seguridad y el procesamiento de consultas. Permite acceder a datos relacionales como parte del procesamiento de XML, así como acceder a datos XML como parte del procesamiento SQL. Entre las funciones de muy alto nivel de XML DB se incluyen:

- Soporte para el lenguaje XQuery (W3C XQuery 1.0 Recommendation).
- Un proceso XSLT que permite realizar transformaciones de XSL dentro de la base de datos.
- Una optimización de reescritura de XPath que puede acelerar las consultas respecto a los datos almacenados en una representación objeto-relacional. Al traducir una expresión usada en XQuery en condiciones directamente sobre las columnas objeto-relacionales, se pueden utilizar los índices regulares de dichas columnas para acelerar el procesamiento de consultas.

28.2.3. Lenguajes procedimentales

Oracle dispone de dos lenguajes procedimentales principales, PL/SQL y Java. PL/SQL fue el lenguaje original de Oracle para los procedimientos almacenados y tiene una sintaxis similar a la del lenguaje Ada. Java se soporta mediante el uso de una máquina virtual de Java dentro del motor de la base de datos. Oracle proporciona un paquete para encapsular procedimientos, funciones y variables relacionadas en una única unidad. Oracle soporta SQLJ (SQL embebido en Java) y JDBC, y proporciona herramientas para generar definiciones de clases de Java correspondientes a los tipos de bases de datos definidos por el usuario.

28.2.4. Dimensiones

El modelado dimensional es una técnica de diseño habitualmente utilizada para los esquemas en estrella relacionales así como para bases de datos multidimensionales. Oracle soporta la creación de **dimensions** (dimensiones) como objetos de metadatos para soportar el procesamiento de consultas contra las bases de datos diseñadas con esta técnica. Los objetos de metadatos se pueden utilizar para guardar información sobre los distintos tipos de atributos de una dimensión, y lo que es quizás más importante, también sobre las relaciones jerárquicas. Consulte la Sección 28.3.10 para ver ejemplos.

28.2.5. OLAP

Oracle proporciona soporte para el procesamiento analítico de bases de datos de varias formas. Además de soportar un rico conjunto de construcciones SQL analíticas (*cube*, *rollup*, conjuntos de agrupaciones, funciones de ventana, etc.), también proporciona almacenamiento multidimensional nativo dentro del servidor de base de datos relacional. Las estructuras de datos multidimensionales permiten el acceso basado en array a los datos y, en las circuns-

tancias apropiadas, este tipo de acceso puede ser mucho más eficiente que los métodos de acceso relacionales tradicionales. El uso de estas estructuras de datos como parte integral de una base de datos relacional proporciona la elección de almacenar los datos en un formato relacional o multidimensional, a la vez que se obtienen las ventajas de las características de Oracle en áreas como la copia de seguridad y la recuperación, la seguridad y las herramientas de administración.

Oracle proporciona contenedores de almacenamiento para datos multidimensionales llamados **espacios de trabajo analíticos.** Un espacio de trabajo analítico contiene los datos dimensionales y las medidas (o hechos) de un cubo OLAP y se almacena dentro de una tabla de Oracle. Desde una perspectiva relacional tradicional, un cubo almacenado en una tabla sería un objeto opaco en el que los datos no se interpretarían directamente en términos de filas y columnas de la tabla. Sin embargo, el servidor OLAP de Oracle en la base de datos tiene el conocimiento para interpretar y acceder a los datos y hace posible dar acceso a SQL como si se hubiesen almacenado en formato de tabla normal. Por tanto, es posible almacenar datos tanto en formato multidimensional como en formato relacional tradicional, dependiendo de cuál sea el óptimo, pudiendo todavía mezclar datos guardados en ambos tipos de representación en una única consulta en SQL. Las vistas materializadas pueden usar cualquiera de las representaciones.

Además de soporte para OLAP de Oracle dentro de la base de datos relacional, el conjunto de productos de Oracle también incluye Essbase, que es una base de datos multidimensional ampliamente utilizada que forma parte de Oracle desde la adquisición de Hyperion.

28.2.6. Disparadores

Oracle proporciona varios tipos de disparadores y varias opciones para el momento y forma en que se invocan (consulte la Sección 5.3 para una introducción a los disparadores en SQL). Los disparadores se pueden escribir en PL/SQL o Java o como llamadas en C.

Para los disparadores que se ejecutan sobre instrucciones LMD, tales como insert, update y delete, Oracle soporta **disparadores de filas (*row triggers*)** y **disparadores de sentencias (*statement triggers*).** Los disparadores de filas se pueden ejecutar una vez por cada fila que se vea afectada (actualización o borrado, por ejemplo) por la operación LMD. Un disparador de sentencias solo se ejecuta una vez por sentencia. En cada caso, el disparador se puede definir como un disparador *before* (antes) o *after* (después), de un **ejemplar**, dependiendo de si se va a invocar antes o después de que se lleve a cabo la operación LMD.

Oracle permite la creación de disparadores **instead of** (en lugar de)para las vistas que no pueden estar sujetas a operaciones LMD. Dependiendo de la definición de la vista, tal vez no sea posible para Oracle traducir sin ambigüedad una sentencia LMD sobre una vista en modificaciones de las tablas de base subyacentes. Por ello, las operaciones LMD sobre vistas están sujetas a numerosas restricciones. Se puede crear un disparador **instead of** sobre una vista para especificar manualmente las operaciones sobre las tablas base que van a producirse en respuesta a la operación LMD sobre la vista. Oracle ejecuta el disparador en lugar de la operación LMD y por consiguiente proporciona un mecanismo de rodeo de las restricciones sobre las operaciones LMD sobre las vistas.

Oracle también tiene disparadores que ejecutan otros eventos, tales como el inicio o finalización de la base de datos, mensajes de error del servidor, inicio o finalización de sesión de un usuario e instrucciones LDD tales como **create, alter** o **drop.**

28.3. Almacenamiento e indexación

En la jerga de Oracle, una **base de datos** consiste en información almacenada en archivos a la cual se accede a través de un **ejemplar,** que es un área de memoria compartida y un conjunto de procesos que interactúan con los datos en los archivos. El **archivo de control** es un pequeño archivo que contiene metadatos de muy alto nivel necesarios para iniciar u operar con un ejemplar. La estructura de almacenamiento de los datos normales y los metadatos se describe en la siguiente sección.

28.3.1. Espacios de tablas

Una base de datos consta de una o más unidades de almacenamiento lógicas denominadas **espacios de tablas.** Cada espacio de tablas, a su vez, consta de una o más estructuras físicas denominadas *archivos de datos*. Estos pueden ser archivos gestionados por el sistema operativo o dispositivos en bruto.

Normalmente una base de datos Oracle tendrá los siguientes espacios de tablas:

- El espacio de tablas del **sistema** y el auxiliar **sysaux,** que siempre se crean. Contienen las tablas del diccionario de datos y almacenamiento para los disparadores y los procedimientos almacenados.
- Espacios de tablas creados para almacenar los datos de usuario. Aunque los datos de usuario se pueden almacenar en el espacio de tablas del **sistema,** frecuentemente es deseable separar los datos de usuario de los datos del sistema. Normalmente la decisión sobre los otros espacios de tablas que se deben crear está basada en el rendimiento, la disponibilidad, la capacidad de mantenimiento y la facilidad de administración. Por ejemplo, puede ser útil tener varios espacios de tablas para las operaciones de copia de seguridad parcial y recuperación.
- El espacio de tablas **undo,** que solamente se usa para almacenar información de deshacer para la gestión y recuperación de transacciones.
- Espacios de tablas **temporales.** Muchas operaciones de base de datos requieren la ordenación de los datos, y esta rutina de ordenación puede tener que almacenar los datos temporalmente en el disco si la ordenación no se puede realizar en la memoria. Se asignan espacios de tablas temporales a la ordenación y asociación para realizar las operaciones de gestión de espacio involucradas en un volcado más eficiente a disco.

Los espacios de tablas también se pueden utilizar como un medio para trasladar datos entre las bases de datos. Por ejemplo, es común trasladar los datos desde un sistema transaccional a un almacén de datos a intervalos regulares. Oracle permite trasladar todos los datos en un espacio de tablas de un sistema a otro sencillamente copiando los archivos y exportando e importando una pequeña cantidad de metadatos del diccionario de datos. Estas operaciones pueden ser mucho más rápidas que descargar los datos de una base de datos y después usar un descargador para insertarlos en la otra. Esta característica de Oracle se denomina **espacios de tablas transportables.**

28.3.2. Segmentos

El espacio en un espacio de tablas se divide en unidades, denominadas **segmentos,** cada una de las cuales contiene datos para una estructura de datos específica. Existen cuatro tipos de segmentos:

- **Segmentos de datos.** Cada tabla en un espacio de tablas tiene su propio segmento de datos donde se almacenan los datos de la tabla a menos que esta esté dividida; si ocurre esto, hay un segmento de datos por división (la división en Oracle se describe en la Sección 28.3.9).

- **Segmentos de índices.** Cada índice en un espacio de tablas tiene su propio segmento de índices, excepto los índices divididos, los cuales tienen un segmento de índices por división.
- **Segmentos temporales.** Son segmentos utilizados cuando una operación de ordenación necesita escribir datos al disco o cuando los datos se insertan en una tabla temporal.
- **Segmentos de deshacer (undo).** Estos segmentos contienen información para deshacer, de forma que se pueda deshacer una transacción no comprometida. Estos segmentos se asignan automáticamente en un espacio de tablas undo especial. También juegan un papel importante en el modelo de control de concurrencia de Oracle, así como para la recuperación de la base de datos, descritos en las Secciones 28.5.1 y 28.5.2. En implementaciones anteriores de la gestión deshacer de Oracle se usaba el término «segmento de retroceso».

Por debajo del nivel de segmento se asigna espacio con un nivel de granularidad denominado **extensión.** Cada extensión consiste en un conjunto de *bloques* contiguos de la base de datos. Un bloque de la base de datos es el nivel más bajo de granularidad en el cual Oracle ejecuta E/S a disco. Un bloque de base de la base de datos no tiene por qué tener el mismo tamaño que un bloque de un sistema operativo, pero debería ser un múltiplo.

Oracle proporciona parámetros de almacenamiento que permiten un control detallado de cómo se asigna y gestiona el espacio, tales como:

- El tamaño de una extensión nueva que se va a asignar para proporcionar espacio a las filas que se insertan en una tabla.
- El porcentaje de utilización de espacio con el cual un bloque de la base de datos se considera lleno y en el cual no se introducirán más filas en ese bloque (dejando algo de espacio libre en un bloque se puede conseguir que las filas existentes aumenten su tamaño cuando se realizan actualizaciones, sin agotar el espacio del bloque).

28.3.3. Tablas

Una tabla estándar en Oracle está organizada en montículo; es decir, la ubicación de almacenamiento de una fila en una tabla no está basada en los valores contenidos en la fila y se fija cuando se inserta. Sin embargo, si la tabla se divide, el contexto de la fila afecta a la partición en la cual está almacenada. Hay varias características y variaciones. Las tablas de montículos se pueden comprimir opcionalmente como se describe en la Sección 28.3.3.2.

Oracle soporta las tablas anidadas; es decir, una tabla puede tener una columna cuyo tipo de datos sea otra tabla. La tabla anidada no se almacena en línea en la tabla padre, sino que se almacena en una tabla separada.

Oracle soporta tablas temporales en las que la duración de los datos es la de la transacción en la cual se insertan los datos o la sesión de usuario. Los datos son privados para la sesión y se eliminan automáticamente al final de su duración.

Una **agrupación** es otra forma de organización de archivos para los datos de la tabla, descrita anteriormente en la Sección 10.6.2, donde se denominaba *agrupación multitabla*. El concepto de «agrupación», en este contexto, no se debería confundir con otros significados de la palabra agrupación, tales como los relacionados con la arquitectura de la computadora. En una organización de archivos en agrupación, las filas de tablas diferentes se almacenan juntas en el mismo bloque según algunas columnas comunes. Por ejemplo, una tabla de departamento y una tabla de empleados se podrían agrupar de forma que cada fila en la tabla departamento se almacene junto con todas las filas de los empleados que trabajan en ese departamento. Los valores de la clave principal o clave externa se utilizan para determinar la ubicación de almacenamiento.

La organización en agrupación implica que una fila pertenece a un lugar específico; por ejemplo, una nueva fila de empleado se debe insertar con las otras filas para el mismo departamento. Por consiguiente, es obligatorio un índice en la columna de agrupación. Una organización alternativa es una **agrupación asociativa.** Aquí, Oracle calcula la localización de una fila aplicando una función asociativa al valor para la columna de agrupación. La función asociativa asigna la fila a un bloque específico en la agrupación asociativa. Puesto que no es necesario el recorrido del índice para acceder a una fila según su valor de columna de agrupación, esta organización puede ahorrar cantidades significativas de E/S a disco.

28.3.3.1. Tablas organizadas con índices

En una **tabla organizada con índices** (*index-organized table*: IOT) los registros se almacenan en un índice de árbol B de Oracle en lugar de en un montículo; esta organización de archivo se describió anteriormente en la Sección 11.4.1, donde se denominó *organización de archivo de árbol* B^+. Una tabla organizada con índices requiere que se identifique una clave única para su uso como la clave del índice. Aunque una entrada en un índice normal contiene el valor de la clave y el identificador de fila de la fila indexada, una tabla organizada con índices reemplaza el identificador de fila con los valores de la columna para el resto de columnas en la tabla. Comparado con el almacenamiento de los datos en una tabla en montículo normal y la creación de un índice según las columnas clave, una tabla organizada con índices puede mejorar el rendimiento y el espacio. Considere la lectura de todos los valores de columna de una fila, dado su valor de clave principal. Para una tabla en montículo se requeriría un examen del índice seguido por un acceso a tabla mediante el identificador de fila. Para una tabla organizada con índices solamente es necesario el examen del índice.

Los índices secundarios sobre columnas que no sean clave de una tabla organizada con índices son distintos de los índices en una tabla en montículo normal. En una tabla en montículo cada fila tiene un identificador de fila fijo que no cambia. Sin embargo, un árbol B se reorganiza al crecer o disminuir cuando se insertan o borran las entradas, y no hay garantía de que una fila permanezca en una ubicación dentro de una tabla organizada con índices. Por ello, un índice secundario en una tabla organizada con índices no contiene identificadores normales de fila, sino **identificadores lógicos de fila.** Un identificador lógico de fila consta de: un identificador de fila física correspondiente al lugar en el que estaba la fila cuando se creó el índice o en la última reconstrucción, y un valor para la clave única. El identificador de fila física se conoce como una «suposición», puesto que sería incorrecto si la fila se ha trasladado. En este caso, la otra parte del identificador lógico de fila, el valor de la clave para la fila, se utiliza para acceder a la fila; sin embargo, este acceso es más lento que si la suposición hubiera sido correcta, puesto que involucra un recorrido del árbol B para la tabla organizada con índices desde la raíz hasta los nodos hoja, incurriendo potencialmente en varias operaciones E/S de disco. Sin embargo, si una tabla es altamente volátil y es probable que un buen porcentaje de suposiciones sean incorrectas, puede ser mejor crear un índice secundario solamente con valores clave (como se describió en la Sección 11.4.1), puesto que el uso de una suposición incorrecta supone una E/S a disco malgastada.

28.3.3.2. Compresión

La característica de compresión de Oracle permite que los datos se almacenen en un formato comprimido que puede reducir drásticamente el espacio necesario para guardar los datos y el número de operaciones de E/S para obtenerlos. El método de compresión

de Oracle es un algoritmo sin pérdida basado en diccionario que comprime cada bloque de forma individual. Toda la información necesaria para descomprimir un bloque se encuentra en el mismo bloque. El algoritmo funciona sustituyendo las apariciones repetidas de un valor en el bloque por punteros a una entrada con dicho valor en una tabla de símbolos (o diccionario) en el bloque. Las entradas se pueden basar en valores repetidos para columnas individuales o para una combinación de columnas.

La compresión de tabla original de Oracle generaba el formato de bloques comprimidos según se cargaban los datos de los bloques en masa en una tabla, y estaba principalmente dirigida a entornos de almacenes de datos (data warehousing). Una nueva característica de compresión OLTP soporta también la compresión en conjunción con las operaciones LMD normales. En este último caso, Oracle comprime los bloques solo después de alcanzar cierto umbral sobre cuantos datos se hayan escrito en el bloque. En este sentido, solo las transacciones que generan que se supere dicho umbral sufrirán una cierta sobrecarga por la compresión del bloque.

28.3.3.3. Seguridad de los datos

Además de las funciones de control de acceso normales, como las contraseñas, los privilegios y los roles de usuario, Oracle dispone de distintas funciones para proteger los datos frente a accesos no autorizados, incluyendo:

- **Cifrado.** Oracle puede almacenar automáticamente los datos de una tabla en formato cifrado, y cifrar y descifrar de forma transparente los datos mediante algoritmos AES o 3DES. El cifrado se puede activar para una base de datos completa o solo para tablas o columnas individuales. El principal motivo de esta función es proteger los datos sensibles fuera del entorno protegido, como cuando se envía una copia de seguridad a una ubicación remota.
- **Valija de base de datos.** Esta función pretende proporcionar una separación de responsabilidades por usuario al acceder a la base de datos. El administrador de la base de datos es un usuario con altos privilegios que normalmente puede hacer casi todo con la base de datos. Sin embargo, puede que no sea apropiado o incluso puede ser ilegal permitir que esa persona acceda a información de datos financieros corporativos o personales de otros empleados. La valija de base de datos incluye un conjunto de mecanismos que se pueden utilizar para restringir o monitorizar el acceso a datos sensibles por parte de un usuario de la base de datos con muchos privilegios.
- **Base de datos privada virtual.** Esta función, descrita en la Sección 9.7.5, permite añadir predicados adicionales automáticamente a las cláusulas **where** de una consulta que accede a determinadas tablas o vistas. Normalmente, la función se usaría de forma que el predicado adicional filtra aquellas filas para las que el usuario no tenga permisos para acceder. Por ejemplo, dos usuarios podrían enviar la misma consulta para buscar la información de todos los empleados de la tabla empleados. Sin embargo, si existe una directiva que limita, para cada usuario, para ver solo la información del empleado que coincide con el ID del usuario, el predicado que se añade automáticamente asegura que las consultas solo devuelven la información del usuario que envió la consulta. Por tanto, los usuarios tienen la impresión de acceder a una base de datos virtual que solo contiene un subconjunto de los datos de la base de datos física.

28.3.4. Índices

Oracle soporta varios tipos distintos de índices. El tipo más comúnmente utilizado es lo que Oracle denomina un índice de árbol B creado en una o varias columnas. Tenga en cuenta que en la terminología de Oracle (y también en otros sistemas de bases de datos) un índice de árbol B es lo que se denomina índice de árbol B^+ en el Capítulo 11. Las entradas de los índices tienen el siguiente formato: para un índice en las columnas col_1, col_2 y col_3, cada fila en la tabla en la que al menos una columna tenga un valor no nulo genera en la entrada de índice:

$$<col_1><col_2><col_3><id\text{-}fila>$$

donde $<col_i>$ indica el valor para la columna i y $<id\text{-}fila>$ es el identificador de fila para la fila. Oracle puede opcionalmente comprimir el prefijo de la entrada para ahorrar espacio. Por ejemplo, si hay muchas combinaciones repetidas de valores $<col_1><col_2>$, la representación de cada prefijo $<col_1><col_2>$ distinto se puede compartir entre las entradas que tienen esa combinación de valores, en lugar de almacenarlo explícitamente para cada entrada. La compresión de prefijos puede suponer sustanciales ahorros de espacio.

28.3.5. Índices de mapas de bits

Los índices de mapas de bits (descritos en la Sección 11.9) utilizan una representación de mapa de bits para entradas de índice que pueden suponer un ahorro sustancial de espacio (y por consiguiente ahorro de E/S a disco), cuando la columna indexada tiene un número moderado de valores distintos. Los índices de mapas de bits en Oracle utilizan la misma clase de estructura de árbol B para almacenar las entradas que un índice normal. Sin embargo, donde un índice normal en una columna tendría entradas de la forma $<col_1><id\text{-}fila>$, una entrada de índice de mapa de bits tiene la forma:

$$<col_1><id\text{-}filainicial><id\text{-}filafinal><mapabits\text{-}comprimido>$$

El mapa de bits conceptualmente representa el espacio de todas las filas posibles en la tabla entre los identificadores de la fila inicial y final. El número de tales filas posibles en un bloque depende de cuántas filas se puedan alojar en un bloque, que es una función del número de columnas en la tabla y sus tipos de datos. Cada bit en el mapa de bits representa una fila posible en un bloque. Si el valor de la columna de esa fila es el de la entrada de índice, el bit se establece a 1. Si la fila tiene algún otro valor o la fila no existe realmente en la tabla, el bit se establece a 0 (es posible que la fila no exista realmente porque un bloque de la tabla puede tener un número más pequeño de filas que el número que se calculó como el máximo posible). Si la diferencia es grande, el resultado puede ser grandes cadenas de ceros consecutivos en el mapa de bits, pero el algoritmo de compresión trata dichas cadenas de ceros, por lo que el efecto negativo es limitado.

El algoritmo de compresión es una variación de una técnica de compresión denominada compresión de mapas de bits alineados (*Byte-Aligned Bitmap Compression*: BBC). Esencialmente, una sección del mapa de bits en la que la distancia entre dos unos consecutivos es suficientemente pequeña se almacena como mapas de bits como tales. Si la distancia entre dos unos es suficientemente grande —es decir, hay un número suficiente de ceros entre ellos— se almacena el número de ceros.

Los índices de mapas de bits permiten que varios índices de la misma tabla se combinen en la misma ruta de acceso si hay varias condiciones sobre las columnas indexadas en la cláusula **where** de una consulta. Los mapas de bits de diferentes índices se obtienen y combinan usando operaciones booleanas correspondientes a las condiciones de la cláusula **where.** Todas las operaciones booleanas se realizan directamente sobre la representación comprimida de los mapas de bits —no es necesaria la descompresión— y el mapa de bits resultante (comprimido) representa las filas que cumplen todas las condiciones lógicas.

La capacidad de utilizar operaciones booleanas para combinar varios índices no se ve limitada a los índices de mapas de bits. Oracle puede convertir identificadores de filas a la representación de mapas de bits comprimidos, por lo que se puede utilizar un índice de árbol B normal en cualquier lugar de un árbol binario u operación de mapa de bits simplemente poniendo un operador id-fila-a-mapa-de-bits en la parte superior del acceso a índices del plan de ejecución.

Como regla nemotécnica, los índices de mapas de bits tienden a ser más eficientes respecto al espacio que los índices de árbol B si el número de valores distintos de la clave es menor que la mitad del número de filas en una tabla. Por ejemplo, en una tabla con un millón de filas, un índice en una columna con menos de 500.000 valores distintos probablemente sería menor si se creara como un índice de mapa de bits. Para las columnas con un número muy pequeño de valores distintos —por ejemplo, las columnas que se refieren a propiedades tales como país, Estado, género, estado marital y varios estados indicadores— un índice de mapas de bits podría requerir solamente una pequeña fracción del espacio normal de un índice de árbol B normal. Cualquier ventaja en el espacio también puede dar lugar a mejoras en el rendimiento en forma de menos operaciones E/S a disco cuando se explora el índice.

28.3.6. Índices basados en funciones

Además de crear índices sobre una o varias columnas de una tabla, Oracle permite crear índices sobre expresiones que involucran a una o más columnas, tales como $col_1 + col_2 * 5$. Por ejemplo, la creación de un índice sobre la expresión *upper*(*nombre*), donde *upper* es una función que devuelve la versión en mayúsculas de una cadena y *nombre* es una columna, es posible realizar búsquedas independientes de la caja (mayúsculas o minúsculas) sobre la columna *nombre*. Con el fin de buscar todas las filas con el nombre «van Gogh» de una forma eficiente se puede utilizar la condición:

$$upper(nombre) = \text{'VAN GOGH'}$$

en la cláusula **where** de la consulta. Oracle entonces casa la condición con la definición de índice y concluye que se puede utilizar el índice para recuperar todas las filas que coincidan con «van Gogh», sin considerar las mayúsculas y minúsculas del nombre cuando se almacenó en la base de datos. Se puede crear un índice basado en una función como un mapa de bits o como un índice de árbol B.

28.3.7. Índices de reunión

Un índice de reunión es un índice en el que las columnas clave no están en la tabla que se referencia mediante los identificadores de filas en el índice. Oracle soporta los índices de reunión de mapas de bits principalmente para su uso con esquemas en estrella (consulte la Sección 20.2.2). Por ejemplo, si hay una columna para los nombres de los productos en una tabla de la dimensión productos, se podría utilizar un índice de reunión de mapas de bits sobre la tabla de hechos con esta columna clave para recuperar las filas de la tabla de hechos que corresponden a un producto con un nombre específico, aunque el nombre no esté almacenado en la tabla de hechos. La forma en la que las filas de las tablas de hechos y de la dimensión se corresponden se basa en una condición de reunión que se especifica cuando se crea el índice y se convierte en parte de los metadatos de índices. Cuando se procesa una consulta, el optimizador buscará la misma condición de reunión en la cláusula **where** de la consulta con el fin de determinar si es aplicable el índice de reunión.

Oracle puede combinar índices de reunión de mapas de bits en una tabla de hechos con otros índices en la misma tabla, sean índices de reunión o no, utilizando los operadores para las operaciones booleanas de mapas de bits.

28.3.8. Índices de dominio

Oracle permite que las tablas sean indexadas por estructuras de índices que no sean propias de Oracle. Esta característica de extensibilidad del servidor Oracle permite a los fabricantes de software desarrollar los llamados **cartuchos** con funcionalidad para dominios de aplicación específicos, tales como texto, datos espaciales e imágenes, con la funcionalidad de indexado más allá de la proporcionada por los tipos de índice estándar de Oracle. Al implementar la lógica para crear, mantener y buscar en el índice, el diseñador de índices debe asegurar que se adhiere a un protocolo específico en su interacción con el servidor Oracle.

Un índice de dominio se debe registrar en el diccionario de datos junto con los operadores que soporta. El optimizador de Oracle considera los índices de dominio como una de las posibles rutas de acceso para una tabla. Oracle permite registrar funciones de coste con los operadores de forma que el optimizador pueda comparar el coste del uso del índice de dominio con el de otras rutas de acceso.

Por ejemplo, un índice de dominio para búsquedas de texto avanzadas puede soportar un operador *contains* (contiene). Una vez que se ha registrado este operador, el índice de dominio se considerará como una ruta de acceso para una consulta como:

select *
from *empleado*
where *contains*(*resumen*, 'LINUX')

donde *resumen* es una columna de texto en la tabla *empleados*. El índice de dominio se puede almacenar en un archivo de datos externo o dentro de una tabla Oracle organizada con índices.

Los índices de dominio se pueden combinar con otros índices (mapas de bits o de árbol B) en la misma ruta de acceso con la conversión entre la representación de mapas de bits y el identificador de fila y usando operaciones booleanas del mapa de bits.

28.3.9. División en particiones

Oracle soporta varias clases de división horizontal de tablas e índices, y esta función juega un papel importante en la capacidad de Oracle a la hora de manejar bases de datos muy grandes. La capacidad de dividir una tabla o índice tiene muchas ventajas en muy diversas áreas.

- La copia de seguridad y recuperación es más sencilla y rápida, puesto que se puede realizar sobre particiones individuales en lugar de sobre toda la tabla.
- Las operaciones de carga en un entorno de almacén de datos son menos intrusivas: se pueden agregar datos a una partición recién creada y después agregar la partición a una tabla, lo que supone una operación instantánea. De igual forma, eliminar una partición con datos obsoletos desde una tabla es muy sencillo en un almacén de datos que mantenga una ventana de datos históricos.
- El rendimiento de la consulta se mejora sustancialmente, puesto que el optimizador puede reconocer que solamente se tiene que acceder a un subconjunto de las particiones de una tabla con el fin de resolver la consulta (poda de particiones). También el optimizador puede reconocer que en una reunión no es necesario intentar hacer corresponder todas las filas de una tabla con todas las filas de la otra, pero que las reuniones solo se deben realizar entre pares coincidentes de divisiones (reunión por particiones).

Un índice de una tabla dividida puede ser un **índice global** o un **índice local.** Las entradas en un índice global se pueden referir a filas de cualquier división. Una tabla con índices locales tiene un

índice físico para cada partición que solo contiene entradas para dicha partición. A no ser que la poda de particiones restrinja una consulta a una única partición, una tabla a la que se acceda con índices locales requerirá muchas pruebas de índices físicos individuales. Sin embargo, un índice local tiene ventajas en entornos de almacenes de datos en los que los nuevos datos se pueden cargar en una nueva partición e indexarlos sin necesidad de mantener índices preexistentes. La carga seguida de la creación de un índice es mucho más eficiente que mantener un índice mientras se realiza la carga. De forma similar, la eliminación de una partición antigua y la parte física de su índice local se puede realizar sin ningún tipo de mantenimiento del índice.

Cada fila en una tabla dividida está asociada con una partición específica. Esta asociación está basada en la columna o columnas de la división que son parte de la definición de una tabla dividida. Hay varias formas para hacer corresponder los valores de columna a divisiones, dando lugar a varios tipos de divisiones, cada una con distintas características: divisiones por rangos, asociativas, por listas y compuestas.

28.3.9.1. División por rangos

En la división por rangos, los criterios de división son rangos de valores. Este tipo de división está especialmente indicado para columnas de fechas, en cuyo caso todas las filas del mismo rango de fechas, digamos un día o un mes, pertenecen a la misma partición. En un almacén de datos en el que los datos se cargan desde sistemas transaccionales a intervalos regulares, la división por rangos se puede utilizar para implementar eficientemente una ventana de datos históricos. Cada carga de datos obtiene su nueva partición propia, haciendo que el proceso de carga sea más rápido y eficiente. El sistema realmente carga los datos en una tabla separada con la misma definición de columna que en una tabla dividida. Se puede entonces verificar la consistencia de los datos, arreglarlos e indexarlos. Después de esto, el sistema puede hacer de la tabla separada una nueva partición mediante un sencillo cambio de los metadatos en el diccionario de datos; una operación casi instantánea.

Mientras no cambien los metadatos, el proceso de carga no afecta a los datos existentes en la tabla dividida en ningún caso. No hay necesidad de realizar ningún mantenimiento de los índices existentes como parte de la carga. Los datos antiguos se pueden eliminar de una tabla sencillamente eliminando su partición; esta operación no afecta al resto de particiones.

Además, en un entorno de almacén de datos las consultas frecuentemente contienen condiciones que los restringen a un cierto periodo de tiempo, tal como una quincena o un mes. Si se utiliza la división de datos por rangos, el optimizador de consulta puede restringir el acceso a los datos de aquellas particiones que son relevantes para la consulta y evitar una exploración de toda la tabla.

Las particiones se pueden crear con conjuntos de puntos explícitos o definidos en base a rangos fijos, como un día o un mes. En el último caso, llamado **división por intervalos,** la creación de la partición se realiza automáticamente cuando se intenta insertar una fila con un valor en un intervalo que no existía anteriormente.

28.3.9.2. División asociativa

En la división asociativa, una función asociativa hace corresponder filas con divisiones según los valores de las columnas de la división. Este tipo de división es útil principalmente cuando es importante distribuir las filas equitativamente entre las particiones o cuando las reuniones por particiones son importantes para el rendimiento de la consulta.

28.3.9.3. División por listas

En la división por listas, los valores asociados con una determinada partición se indican en una lista. Este tipo de partición es útil cuando los datos en la columna de la división tienen un conjunto relativamente pequeño de valores discretos. Por ejemplo, una tabla con una columna de Estado se puede dividir implícitamente por región geográfica si cada lista de división contiene los Estados que pertenecen a la misma región.

28.3.9.4. División compuesta

En la división compuesta la tabla se divide por rangos, intervalos o listas, pero cada partición tiene subparticiones mediante el uso de la división por intervalos, asociativa o por listas. Por ejemplo, una tabla se puede dividir por intervalos sobre una columna de fechas y subdividir de forma asociativa sobre una columna que se usa frecuentemente como columna de reunión. Las subparticiones permiten utilizar reuniones sobre particiones cuando la tabla se reúne con otras.

28.3.9.5. División por referencias

En la división por referencias, la clave de la división se resuelve de acuerdo con la restricción de clave externa con otra tabla. La dependencia entre las tablas permite que las operaciones de mantenimiento actúen en cascada automáticamente.

28.3.10. Vistas materializadas

La característica de vista materializada (consulte la Sección 4.2.3) permite almacenar el resultado de una consulta SQL y utilizarlo en un procesamiento posterior. Además, Oracle mantiene el resultado materializado, actualizándolo cuando se actualizan las tablas a las que se hicieron referencia en la consulta. Las vistas materializadas se utilizan en el almacén de datos para acelerar el procesamiento de la consulta, pero esta tecnología también se usa para la réplica en entornos distribuidos y móviles.

En el almacén de datos, un uso común de las vistas materializadas consiste en resumir los datos. Por ejemplo, un tipo común de consulta solicita «la suma de las ventas de cada cuatrimestre durante los últimos dos años». El precálculo de los resultados, o algún resultado parcial, de dicha consulta puede acelerar drásticamente el procesamiento de la consulta comparado con el cálculo desde cero con la agregación de todos los registros de ventas por detalle.

Oracle soporta reescrituras automáticas de las consultas que aprovechan cualquier vista materializada útil cuando se resuelve una consulta. La reescritura consiste en cambiar la consulta para utilizar la vista materializada en lugar de las tablas originales en la consulta. Además, la reescritura puede agregar reuniones adicionales o el procesamiento de agregación si son necesarios para obtener un resultado correcto. Por ejemplo, si una consulta necesita las ventas por cuatrimestre, la reescritura puede aprovechar una vista que materializa las ventas por mes, añadiendo la agregación adicional para agrupar los meses en cuatrimestres. Oracle tiene un tipo de objeto de metadatos denominado dimensión que permite definir relaciones jerárquicas en las tablas. Por ejemplo, para una tabla de la dimensión temporal en un esquema en estrella, Oracle puede definir un objeto de metadatos *dimension* para especificar cómo se agrupan los días en meses, los meses en cuatrimestres, los cuatrimestres en años, y así sucesivamente. De igual forma, se pueden especificar las propiedades jerárquicas relacionadas con la geografía; por ejemplo, cómo los distritos de ventas se agrupan en regiones. La lógica de la reescritura de la consulta examina estas relaciones, puesto que permite utilizar una vista materializada para clases más amplias de consultas.

El objeto contenedor para una vista materializada es una tabla, lo que significa que una vista materializada se puede indexar, dividir o estar sujeta a otros controles para mejorar el rendimiento de la consulta.

Cuando hay cambios en los datos de las tablas referenciadas en la consulta que define una vista materializada, se debe actualizar la vista materializada para reflejar dichos cambios. Oracle soporta tanto la actualización completa de una vista materializada como una actualización rápida incremental. En una actualización completa, Oracle vuelve a calcular la vista materializada desde cero, lo cual puede ser la mejor opción si las tablas subyacentes han tenido cambios significativos, por ejemplo, debidos a una carga masiva. En una actualización incremental, Oracle actualiza la vista utilizando registros que fueron cambiados en las tablas subyacentes. La actualización de la vista se puede ejecutar *por compromiso* como parte de la transacción que cambia las tablas subyacentes o, más adelante en el tiempo, *a demanda*. La actualización incremental puede ser mejor si el número de filas que se han cambiado es pequeño. Hay algunas restricciones sobre las clases de consultas para las que una vista materializada se puede actualizar de forma incremental (y otras que indican si se puede crear siquiera una vista materializada).

Una vista materializada es similar a un índice en el sentido de que, aunque puede mejorar el rendimiento de la consulta, usa espacio, y su creación y mantenimiento consumen recursos. Para ayudar a resolver este compromiso Oracle proporciona un asesor que puede ayudar al usuario a crear vistas materializadas menos costosas, dada una carga de trabajo particular como entrada.

28.4. Procesamiento y optimización de consultas

Oracle soporta una gran variedad de técnicas de procesamiento en su motor de procesamiento de consultas. Algunas de las más importantes se describen aquí brevemente.

28.4.1. Métodos de ejecución

Se puede acceder a los datos mediante una serie de métodos de acceso:

- **Exploración de tabla completa.** El procesador de consultas explora toda la tabla, obtiene información sobre los bloques que forman la tabla del mapa de extensión y explora estos bloques.
- **Exploración de índices.** El procesador crea una clave de comienzo y/o finalización a partir de las condiciones en la consulta y la utiliza para explorar una parte relevante del índice. Si hay columnas que se tienen que recuperar, y que no son parte del índice, la exploración del índice irá seguida de un acceso a la tabla mediante el índice del identificador de fila. Si no hay disponible ninguna clave de inicio o parada, la exploración será una exploración de índice completa.
- **Exploración rápida completa de índices.** El procesador explora las extensiones de la misma forma que la extensión de tabla en una exploración de tabla completa. Si el índice contiene todas las columnas de la tabla que se necesitan en ella y no hay buenas claves de inicio y parada que puedan reducir significativamente esa porción del índice que se exploraría en una exploración de índices normal, este método puede ser la forma más rápida de acceder a los datos. Esto es así porque la exploración rápida completa aprovecha de forma total la E/S de disco de varios bloques. Sin embargo, a diferencia de una exploración completa normal, que recorre los bloques hoja del índice en orden, una exploración rápida completa no garantiza que la salida preserve el orden del índice.
- **Reunión de índices.** Si una consulta necesita solamente un pequeño subconjunto de columnas de una tabla ancha, pero ningún índice contiene todas estas columnas, el procesador puede utilizar una reunión de índices para generar la información relevante sin acceder a la tabla, reuniendo varios índices que contienen en conjunto las columnas necesarias. Ejecuta las reuniones como reunión por asociación sobre los identificadores de filas desde los distintos índices.
- **Acceso a agrupaciones y agrupaciones asociadas.** El procesador accede a los datos utilizando la clave de agrupación.

Oracle cuenta con diversas formas de combinar información desde varios índices en una única ruta de acceso. Esta posibilidad permite utilizar conjuntamente varias condiciones en la cláusula **where** para calcular el conjunto de resultados de la forma más eficientemente posible. La funcionalidad incluye la capacidad de ejecutar las operaciones booleanas conjunción **(and),** disyunción **(or)** y diferencia **(minus)** sobre mapas de bits que representan los identificadores de filas. Hay también operadores que hacen corresponder una lista de identificadores de filas con mapas de bits y viceversa, lo que permite que los índices de árbol B normales y los índices de mapas de bits utilicen la misma ruta de acceso. Además, para muchas consultas que involucran **count**(*) en selecciones sobre una tabla el resultado se puede calcular simplemente contando los bits activados en el mapa de bits generado mediante la aplicación de las condiciones de la cláusula **where** sin acceder a la tabla.

Oracle soporta varios tipos de reuniones en el motor de ejecución: reuniones internas, externas, semirreuniones y antirreuniones (una antirreunión en Oracle devuelve las filas de la parte izquierda de la entrada que no coinciden con ninguna fila en la parte derecha de la entrada; esta operación se denomina antisemirreunión en otros libros). Evalúa cada tipo de reunión mediante uno de los tres métodos: reunión por asociación, reunión por mezcla-ordenación o reunión en bucle anidado.

28.4.2. Optimización

En el Capítulo 13 se ha estudiado el tema general de la optimización de consultas. Aquí se trata la optimización en el contexto de Oracle.

28.4.2.1. Transformaciones de consultas

Oracle realiza la optimización de consultas en varios pasos. Uno consiste en realizar varias transformaciones de las consultas y reescrituras que fundamentalmente cambian la estructura de la consulta. Otro paso es realizar la selección de la ruta de acceso para determinar las rutas, los métodos de reunión y el orden de reunión. Dado que no todas las técnicas de transformación tienen garantizado su beneficio, Oracle realiza transformaciones basadas en el coste en donde entrelazan tales transformaciones y la selección de accesos. Para cada transformación generada se realiza una selección de ruta de acceso y se genera una estimación del coste, aceptándose o rechazándose las transformación dependiendo del coste del plan de ejecución resultante.

Algunos de los principales tipos de transformaciones y reescrituras soportados por Oracle son los siguientes:

- **Mezcla de vistas.** La referencia de la vista en una consulta es reemplazada por la definición de la vista. Esta transformación no es aplicable a todas las vistas.
- **Mezcla compleja de vistas.** Oracle ofrece esta característica para ciertas clases de vistas que no están sujetas a la mezcla normal de vistas puesto que tienen un **group by** o **select distinct** en su definición. Si dicha vista se combina con otras tablas, Oracle puede conmutar las reuniones y las operaciones de ordenación y asociación utilizadas por **group by** o **distinct.**

- **Subconsultas planas.** Oracle tiene una serie de transformaciones que convierten varias clases de subconsultas en reuniones, semirreuniones o antirreuniones. Esta conversión también se denomina *descorrelación* y se describe brevemente en la Sección 13.4.4.
- **Reescritura de vistas materializadas.** Oracle tiene la capacidad de reescribir una consulta automáticamente para aprovechar las vistas materializadas. Si alguna parte de la consulta se puede casar con una vista materializada existente, Oracle puede remplazar esta parte de la consulta con una referencia a la tabla en la cual la vista está materializada. Si es necesario, Oracle agrega condiciones de reunión u operaciones **group by** para preservar la semántica de la consulta. Si son aplicables varias vistas materializadas, Oracle escoge la que reduce en mayor medida la cantidad de datos que se tienen que procesar. Además, Oracle somete la consulta reescrita y la versión original al proceso completo de optimización, produciendo un plan de ejecución y un coste asociado estimado para cada una. Oracle entonces decide si ejecutar la versión original de la consulta o la reescrita según la estimación del coste.
- **Transformación en estrella.** Oracle posee una técnica especial para evaluar las consultas en esquemas en estrella, conocidas como transformación en estrella. Cuando una consulta contiene una reunión de una tabla de hechos con tablas dimensionales y selecciones sobre los atributos de las tablas dimensionales, la consulta se transforma eliminando la condición de la reunión entre la tabla de hechos y las tablas dimensionales y remplazando la condición de selección en cada tabla dimensional por una subconsulta de la forma:

  tabla_de_hechos$_i$ **in**
  (**select** cp **from** *tabla_dimensional*$_i$
  where <condiciones sobre
  tabla_dimensional$_i$>)

  Se genera dicha subconsulta para cada tabla dimensional que tiene algún predicado restrictivo. Si la dimensión tiene un esquema en copo de nieve (consulte la Sección 20.2) la subconsulta contendrá una reunión de las tablas aplicables que forman la dimensión.

  Oracle utiliza los valores que son devueltos desde cada subconsulta para probar un índice sobre la columna de la tabla de hechos correspondiente, obteniendo un mapa de bits como resultado. Los mapas de bits generados desde distintas subconsultas se combinan con una operación **and** de mapas de bits. El mapa de bits resultante se puede utilizar para acceder a las filas de las tablas de hechos coincidentes. Por ello, solamente se accederá a las filas en la tabla de hechos que coinciden simultáneamente en las condiciones de las dimensiones restringidas. Tanto la decisión sobre si usar una subconsulta para una dimensión en particular es eficiente en coste, como la decisión de si la consulta reescrita es mejor que la original se basan en las estimaciones de coste del optimizador.

28.4.2.2. Selección de la ruta de acceso

Oracle tiene un optimizador basado en el coste que determina el orden de la reunión, los métodos de reunión y las rutas de acceso. Para cada operación en que el optimizador considera una función de coste asociada, el optimizador intenta generar la combinación de operaciones que tiene el menor coste global.

Para estimar el coste de una operación, el optimizador se basa en las estadísticas que se han calculado para los objetos del esquema tales como tablas e índices. Las estadísticas contienen información sobre el tamaño del objeto, la cardinalidad, la distribución de datos de las columnas de la tabla, y datos similares. Para las estadísticas de columnas, Oracle soporta histogramas equilibrados en altura e histogramas de frecuencia. Los histogramas equilibrados en altura también se denominan *histogramas en equiprofundidad* y se describen en la Sección 13.3.1.

Para facilitar la recogida de las estadísticas del optimizador, Oracle puede supervisar la actividad de la modificación sobre tablas y sigue la pista de aquellas tablas que han sido objeto de suficientes cambios como para que pueda ser apropiado un nuevo cálculo de las estadísticas. Oracle también sigue las columnas que se utilizan en las cláusulas **where** de las consultas, lo que hace que sean potenciales candidatas para la creación del histograma. Con un solo comando es posible que Oracle actualice las estadísticas de las tablas que se han modificado sustancialmente. Oracle utiliza un muestreo para acelerar el proceso de recoger la nueva estadística y elige de forma automática el menor porcentaje de la muestra que sea adecuado. También determina si la distribución de las columnas marcadas merece la creación de histogramas; si la distribución está cerca de ser uniforme, Oracle utiliza una representación más sencilla de las estadísticas de columnas.

En algunos casos puede ser imposible que el optimizador estime de forma precisa la selectividad de una condición en la cláusula **where** de una consulta simplemente a partir de las estadísticas almacenadas. Por ejemplo, la condición puede ser una expresión sobre una sola columna, como $f(col + 3) > 5$. Otro caso de consultas problemáticas es cuando se tienen varios predicados sobre columnas correlacionadas de alguna forma. Puede ser difícil determinar la selectividad combinada de estos predicados. Oracle trata estos problemas con el *muestreo dinámico*. El optimizador puede muestrear aleatoriamente una pequeña parte de la tabla y aplicar todos los predicados relevantes a la muestra para determinar el porcentaje de filas que los cumplen. Esta característica también puede manejar tablas temporales en las que la amplitud y visibilidad de los datos puede evitar la recogida de estadísticas regulares.

Oracle utiliza tanto el coste de CPU como de E/S en disco en el modelo de coste en el optimizador. Para equilibrar los dos componentes almacena las medidas sobre la velocidad de CPU y rendimiento de E/S de disco como parte de la estadística del optimizador. El paquete de Oracle para recoger la estadística del optimizador calcula estas medidas.

Para aquellas consultas que involucran un número no trivial de reuniones, el espacio de búsqueda es un problema para el optimizador de consultas. Oracle soluciona este tema de varias formas. El optimizador genera un orden inicial de la reunión y después decide sobre los mejores métodos de reunión y sobre las rutas de acceso para ese orden de reunión. A continuación cambia el orden de las tablas y determina los mejores métodos de reunión y rutas de acceso para el nuevo orden, y así sucesivamente, guardando el mejor plan encontrado hasta entonces. Oracle mantiene reducida la optimización si el número de los distintos órdenes de la reunión que se han considerado es tan grande que el tiempo gastado por el optimizador puede ser elevado comparado con el que se gastaría para ejecutar el mejor plan encontrado hasta entonces. Puesto que este corte depende del coste estimado para el mejor plan encontrado hasta entonces, es importante encontrar pronto un buen plan, de forma que el optimizador se pueda parar después de un pequeño número de órdenes de la reunión, dando lugar a un mejor tiempo de respuesta. Oracle utiliza varias heurísticas para el orden inicial y para aumentar la probabilidad de que el primer orden de reunión se considere bueno.

Por cada orden de reunión que se considera, el optimizador puede hacer pasadas adicionales por las tablas para decidir los métodos de reunión y las rutas de acceso. Tales pasadas adicionales capturarían efectos globales laterales específicos sobre la selección de la ruta de acceso. Por ejemplo, una combinación específica de métodos de reunión y rutas de acceso puede eliminar la nece-

sidad de ejecutar una ordenación **order by.** Puesto que tal efecto lateral global puede no ser obvio cuando se consideran localmente los costes de los distintos métodos de reunión y de rutas de acceso, se utiliza una pasada separada que capture un efecto lateral específico para encontrar un posible plan de ejecución con un mejor coste conjunto.

28.4.2.3. Poda de particiones

Para las tablas divididas, el optimizador intenta casar las condiciones de la cláusula **where** de una consulta con los criterios de partición de la tabla, para evitar acceder a particiones que no son necesarias para el resultado. Por ejemplo, si una tabla se divide por intervalos de fechas y la consulta se restringe a datos entre dos fechas dadas, el optimizador determina qué particiones contienen los datos entre las fechas especificadas y asegura que solo se accede a dichas particiones. Este escenario es muy común y mejora sustancialmente la velocidad si solo se necesitan un subconjunto de particiones.

28.4.2.4. SQL Tuning Advisor

Además del proceso normal de optimización, el optimizador de Oracle se puede usar en modo de ajuste como parte del asesor de ajuste de SQL (*SQL Tuning Advisor*) para generar planes de ejecución más eficientes de los que normalmente se generarían. Esta característica es especialmente útil para las aplicaciones empaquetadas que generan el mismo conjunto de sentencias repetidamente, por lo que el esfuerzo de ajustar estas sentencias por su rendimiento puede suponer ventajas futuras.

Oracle supervisa la actividad de la base de datos y almacena automáticamente información sobre instrucciones SQL de gran coste en un repositorio de carga de trabajo; consulte la Sección 28.8.2. Las sentencias SQL de carga elevada son las que usan más recursos porque se ejecutan gran número de veces o porque son inherentemente costosas. Estas sentencias son candidatas lógicas para su ajuste, ya que el impacto sobre el sistema es mayor. El *SQL Tuning Advisor* se puede usar para mejorar el rendimiento de estas sentencias sugiriendo recomendaciones que se catalogan en las siguientes categorías:

- **Análisis estadístico.** Oracle comprueba si están disponibles y actualizadas las estadísticas necesarias para el optimizador y proporciona recomendaciones para recopilarlas.
- **Análisis de rendimiento de SQL.** El perfil de una sentencia SQL es un conjunto de informaciones para ayudar al optimizador a tomar mejores decisiones la próxima vez que se optimice la instrucción. El optimizador puede generar a veces planes de ejecución ineficientes si no es capaz de estimar de forma precisa las cardinalidades y las selectividades, lo cual puede ocurrir como resultado de una correlación de datos o del uso de ciertos tipos de constructores. Al ejecutar el optimizador en modo de ajuste para generar un perfil o análisis de rendimiento, el optimizador intenta comprobar que sus suposiciones son correctas según el muestreo dinámico y la evaluación parcial de la sentencia SQL. Si encuentra pasos en el proceso de optimización en los que las suposiciones del optimizador son incorrectas, genera un factor de corrección para cada paso que será parte del perfil. La optimización en el modo de ajuste puede consumir mucho tiempo, pero puede ser útil si el perfil mejora significativamente el rendimiento de la sentencia. Si se crea un perfil, este se almacena de forma persistente y se usa cada vez que se optimice la instrucción. Los perfiles se pueden usar para ajustar las instrucciones SQL sin cambiar su forma textual, lo cual es importante porque a menudo es imposible que el administrador de bases de datos modifique las instrucciones generadas por las aplicaciones.
- **Análisis de la ruta de acceso.** Oracle sugiere la creación de índices adicionales que podrían acelerar la ejecución de la sentencia usando el análisis del optimizador.
- **Análisis de la estructura de SQL.** Oracle sugiere cambios en la estructura de las sentencias SQL que permitirían una ejecución más eficiente.

28.4.2.5. Gestión del plan de SQL

Las aplicaciones empaquetadas suelen generar un gran número de sentencias SQL que se ejecutan repetidamente. Si la aplicación se comporta adecuadamente, es habitual que los administradores de la base de datos no deseen realizar cambios en el comportamiento de la misma. Si los cambios generan un mejor rendimiento, existe una leve mejora percibida, pues el rendimiento ya era suficientemente bueno. Por otra parte, si el cambio genera una degradación del rendimiento, puede perjudicar a la aplicación si una consulta crítica deteriora el tiempo de respuesta a límites inaceptables.

Un ejemplo de cambio del comportamiento es un cambio en el plan de ejecución de la consulta. Este cambio puede reflejar de forma legítima cambios en las propiedades de los datos, como que una tabla crezca excesivamente, pero el cambio puede deberse a consecuencias no intencionadas de otras acciones, como un cambio en las rutinas de obtención de estadísticas del optimizador o una actualización a una nueva versión del RDBMS con un nuevo comportamiento del optimizador.

La característica de la gestión del plan de SQL de Oracle trata los riesgos asociados con los cambios en la ejecución de planes, manteniendo un conjunto de planes de ejecución confiables para una cierta carga de trabajo y realizando cambios en dichos panes solo después de haberse comprobado que no generan degradación del comportamiento. Esta característica tiene tres elementos principales:

1. **Captura de planes base de SQL.** Oracle puede capturar planes de ejecución para una carga de trabajo y guardar un histórico de planes de cada sentencia SQL. El plan base es un conjunto de planes para una carga de trabajo con unas características de rendimiento fiables contra los que se pueden comparar futuros cambios de planes. Una sentencia podría tener más de un plan base.
2. **Selección de planes base de SQL.** Una vez que el optimizador genera un plan para una sentencia SQL, comprueba si existe un plan base para tal sentencia. Si la sentencia existe ya pero el nuevo plan es diferente de cualquiera previo, se utilizará aquel plan que el optimizador considere como óptimo. El nuevo plan generado se añadirá al histórico de planes para la sentencia y podría formar parte de una base futura.
3. **Evolución del plan base de SQL.** Periódicamente es conveniente intentar generar nuevos planes de ejecución como parte de los planes confiables en la base. Oracle soporta añadir nuevos planes a la base con o sin *verificación*. Si se elige la opción de verificación, Oracle ejecutará el nuevo plan generado y comparará su rendimiento con la base para asegurarse de que no genera problemas de rendimiento.

28.4.3. Ejecución en paralelo

Oracle permite ejecutar en paralelo una única sentencia SQL mediante la división del trabajo entre varios procesos en una computadora multiproceso. Esta característica es especialmente útil para operaciones intensivas en cómputo que de otra forma se ejecutarían en un tiempo inaceptablemente largo. Ejemplos representativos son las consultas de apoyo para la toma de decisiones que necesitan procesar grandes cantidades de datos, las cargas de datos en un almacén de datos y la creación o reconstrucción de índices.

Con el fin de lograr una buena aceleración mediante el paralelismo es importante que el trabajo involucrado en la ejecución de la instrucción se divida en gránulos que se pueden procesar de forma independiente mediante los distintos procesadores en paralelo. Dependiendo del tipo de operación, Oracle tiene diversas formas de dividir el trabajo.

Para operaciones que acceden a objetos base (tablas e índices), Oracle puede dividir el trabajo mediante segmentos horizontales de datos. Para algunas operaciones, tales como una exploración completa de una tabla, cada uno de dichos segmentos puede ser un intervalo de bloques; cada proceso de consulta en paralelo explora la tabla desde el bloque al comienzo del intervalo hasta el final. Para otras operaciones en una tabla dividida, como la exploración de un intervalo de índices, el segmento sería una partición. El paralelismo basado en intervalos de bloques es más flexible ya que estos se pueden determinar dinámicamente según una variedad de criterios y no quedan atados a la definición de la tabla.

Las reuniones en paralelo se pueden realizar de distintas formas. Una forma consiste en dividir una de las entradas a la reunión entre procesos paralelos y permitir que cada proceso reúna su segmento con la otra entrada de la reunión; este es el método de reunión con fragmentos y réplicas de la Sección 18.5.2.2. Por ejemplo, si una tabla grande se reúne con una pequeña mediante una reunión por asociación, Oracle divide la tabla grande entre los procesos y envía una copia de la tabla pequeña a cada proceso, la cual a su vez reúne su segmento con la tabla menor. Si ambas tablas son grandes sería prohibitivamente costoso enviar una de ellas a todos los procesos. En ese caso Oracle logra el paralelismo mediante la división de los datos entre los procesos con la asociación de los valores de las columnas de la reunión (es el método de reunión por asociación dividida de la Sección 18.5.2.1). Cada tabla se explora en paralelo mediante un conjunto de procesos y cada fila en la salida se pasa a un proceso de un conjunto de procesos que van a ejecutar la reunión. El proceso que obtiene la fila se determina mediante una función de asociación sobre los valores de la columna de reunión. Por ello, cada proceso de reunión solo obtiene las filas que podrían potencialmente coincidir y las filas correspondientes que no podrían ir a parar a procesos diferentes.

Oracle organiza en paralelo las operaciones de ordenación mediante los rangos de valores de la columna en la cual se ejecuta la ordenación (es decir, usando la ordenación de división por rangos de la Sección 18.5.1). A cada proceso que participa en la ordenación se le envían filas con los valores de ese rango y ordena las filas en su rango. Para maximizar las ventajas del paralelismo las filas se tienen que dividir lo más equitativamente posible entre los procesos en paralelo, pero entonces surge el problema de determinar las fronteras de rango que generan una buena distribución. Oracle soluciona el problema mediante un muestreo dinámico de un subconjunto de las filas en la entrada a la ordenación antes de decidir las fronteras del rango.

28.4.3.1. Estructura de procesos

Los procesos involucrados en la ejecución en paralelo de una sentencia SQL consisten en un proceso coordinador y una serie de procesos servidores en paralelo. El coordinador es responsable de asignar trabajos a los servidores en paralelo y de recoger y devolver los datos a los procesos del usuario que enviaron la sentencia. El grado de paralelismo es el número de procesos servidores en paralelo que se asignan para ejecutar una operación primitiva como parte de la instrucción. El grado de paralelismo lo determina el optimizador, pero se puede reducir dinámicamente si aumenta la carga en el sistema.

Los servidores en paralelo operan con un modelo productor/consumidor. Cuando es necesaria una secuencia de operaciones para procesar una sentencia, el conjunto productor de servidores ejecuta la primera operación y pasa los datos resultantes al conjunto de consumidores. Por ejemplo, si una exploración de tabla completa es seguida por una ordenación y el grado de paralelismo es 32, habría 32 servidores productores que ejecutan la exploración de la tabla y pasan el resultado a 32 servidores consumidores que ejecutan la ordenación. Si es necesaria una operación posterior, como otra ordenación, las funciones de los dos conjuntos de servidores se intercambian. Los servidores que originalmente ejecutaban la exploración de la tabla adoptan la función de consumidores de la salida producida por la primera ordenación y la utilizan para ejecutar la segunda ordenación. Por ello se realiza una secuencia de operaciones pasando los datos entre dos conjuntos de servidores que alternan sus funciones como productores y consumidores. Los servidores se comunican entre sí mediante las memorias intermedias sobre hardware de memoria compartida y mediante las conexiones de red de alta velocidad sobre configuraciones MPP (sin compartimiento) y los sistemas agrupados (discos compartidos).

Para sistemas sin compartimiento, el coste para acceder a los datos del disco no es uniforme entre los procesos. Un proceso que se ejecuta en un nodo que tiene acceso directo a un dispositivo puede procesar los datos sobre ese dispositivo más rápidamente que un proceso que tiene que recuperar los datos a través de la red. Oracle utiliza el conocimiento sobre la afinidad dispositivo a nodo y dispositivo a proceso —es decir, la capacidad de acceder a los dispositivos directamente— cuando distribuye el trabajo entre servidores en ejecución paralela.

28.4.4. Caché de resultados

La característica de caché de resultados de Oracle permite que los resultados de una consulta o bloque de consultas (es decir, una vista referenciada en una consulta) se ubiquen en caché en la memoria y se reutilicen si se vuelve a ejecutar de nuevo. Las actualizaciones en las tablas subyacentes invalidan los resultados de la caché, por lo que esta característica funciona mejor para consultas en tablas que son relativamente estáticas y en las que los conjuntos de resultados son relativamente pequeños. Considere como ejemplo una parte de una página web que se guarda en la base de datos y que no cambia con frecuencia pese a lo habitual de su acceso. Para este tipo de aplicación, la caché de resultados sería una alternativa más ligera que usar vistas materializadas, que requieren la creación explícita y la administración de nuevos objetos persistentes en la base de datos.

28.5. Control de concurrencia y recuperación

Oracle soporta técnicas de control de concurrencia y recuperación que proporcionan una serie de características útiles.

28.5.1. Control de concurrencia

El control de concurrencia multiversión de Oracle se basa en el protocolo de aislamiento de instantáneas descrito en la Sección 15.7. Para las consultas de solo lectura se proporcionan instantáneas consistentes en lectura, que son vistas de la base de datos tal como existía en un cierto momento, y que contienen todas las actualizaciones que se comprometieron hasta ese momento pero no el resto. Por ello, no se utilizan los bloqueos de lectura y las consultas de solo lectura no interfieren con otra actividad de la base de datos en términos de bloqueos.

Oracle soporta la consistencia de lectura en un nivel de instrucción y de transacción: al comienzo de la ejecución de una instrucción o una transacción (dependiendo del nivel de consistencia que se utilice), Oracle determina el número de cambio del sistema (*system change number*: SCN) actual. El SCN esencialmente actúa como una marca temporal en la que el tiempo se mide en términos de compromisos de la base de datos, en lugar del tiempo de reloj.

Si en el transcurso de una consulta se determina que un bloque de datos tiene un SCN mayor que el que se está asociando a la consulta, es evidente que se ha modificado el bloque de datos después del SCN de la consulta original mediante alguna otra transacción y puede o no haberse comprometido. Por ello, los datos del bloque no se pueden incluir en una vista consistente de la base de datos como existía a la hora del SCN de la consulta. En su lugar, se debe utilizar una versión anterior de los datos en el bloque, en concreto el que tenga el mayor SCN que no exceda el SCN de la consulta. Oracle recupera la versión de los datos desde el segmento de retroceso (los segmentos de retroceso se describen en la Sección 28.5.2). Por esta razón, suponiendo que el espacio de retroceso es lo suficientemente grande, Oracle puede devolver un resultado consistente de la consulta incluso aunque los datos se hayan modificado varias veces desde que comenzara la ejecución de la consulta. Si el bloque con el SCN deseado ya no existe en el segmento de retroceso, la consulta devolverá un error. Habría una indicación de que el segmento de retroceso no se ha dimensionado adecuadamente, dada la actividad del sistema.

En el modelo de concurrencia de Oracle, las operaciones de lectura no bloquean las operaciones de escritura y las operaciones de escritura no bloquean las operaciones de lectura, una propiedad que permite un alto grado de concurrencia. En concreto, el esquema permite que se ejecuten consultas largas (por ejemplo, consultas de informes) en un sistema con una gran cantidad de actividad transaccional. Esta clase de escenario es normalmente problemático para sistemas de bases de datos en los que las consultas utilizan bloqueos de lectura, puesto que la consulta puede fallar al adquirirlos o bloquear grandes cantidades de datos por mucho tiempo evitando, por consiguiente, la actividad transaccional de los datos y reduciendo la concurrencia. (Una alternativa que se emplea en algunos sistemas es utilizar un grado inferior de consistencia, tal como la consistencia en grado dos, pero eso podría producir resultados inconsistentes en la consulta).

El modelo de concurrencia de Oracle se utiliza como base para las características **flashback.** Estas características permiten a los usuarios establecer un cierto número SCN o tiempo de reloj en su sesión y ejecutar operaciones sobre los datos que existían en dicho momento (supuesto que los datos todavía existían en el segmento de retroceso). Normalmente, en un sistema de bases de datos, una vez que se ha comprometido un cambio no hay forma de retroceder al estado anterior de los datos a menos que se realicen restauraciones desde copias de seguridad. Sin embargo, la recuperación de una base de datos muy grande puede ser muy costosa, especialmente si el objetivo es solamente recuperar algunos datos que un usuario ha borrado inadvertidamente. Las características de flashback proporcionan un mecanismo mucho más sencillo para tratar los errores del usuario. Entre ellas se encuentra la posibilidad de restaurar una tabla o una base de datos completa a un momento pasado sin recuperación desde las copias de seguridad, la posibilidad de realizar consultas sobre los datos que existieron en algún momento anterior, la posibilidad de realizar un seguimiento del cambio de una o más filas a lo largo del tiempo, así como la posibilidad de examinar los cambios de la base de datos en el nivel de transacciones.

Puede resultar deseable ser capaz de realizar un seguimiento de los cambios en las tablas más allá de lo posible mediante la retención de segmentos de retroceso normales. (Por ejemplo, las regulaciones gubernamentales pueden requerir que dichos cambios sean trazables durante un determinado número de años). Para ello se puede trazar una tabla mediante la característica de **archivo de flashback** que crea una versión histórica, interna, de la tabla. Un proceso en segundo plano convierte la información de retroceso en entradas en la tabla histórica, que se puede utilizar para proporcionar funcionalidad de flashback para periodos de tiempo arbitrarios.

Oracle soporta dos niveles de aislamiento ANSI/ISO: **lectura comprometida** y **secuenciable.** No hay soporte para lecturas sucias puesto que no hay necesidad. La consistencia de lectura en el nivel de sentencia se corresponde con el nivel de aislamiento lectura comprometida, mientras que el nivel de aislamiento en el nivel de transacción se corresponde con el nivel de aislamiento secuenciable. El nivel de aislamiento se puede establecer para una sesión o para una transacción individual. La consistencia de lectura en el nivel de sentencia (es decir, lectura comprometida) es el nivel de aislamiento predeterminado.

Oracle utiliza un bloqueo en el nivel de las filas. Las actualizaciones de distintas filas no entran en conflicto. Si dos escritores intentan modificar la misma fila, uno espera hasta que el otro comprometa o retroceda, y entonces puede devolver un error de conflicto de escritura o seguir y modificar la fila; los errores de conflicto de escritura se detectan a partir de la versión *primera actualización gana* del aislamiento de instantáneas, descrita en la Sección 15.7. (En la Sección 15.7 se describen también ciertos casos de ejecución no secuenciable que pueden producirse con el aislamiento de instantánea, y técnicas para evitar este problema). Los bloqueos se mantienen mientras dura la transacción.

Además de los bloqueos en el nivel de las filas que evitan las inconsistencias debidas a la actividad LMD, Oracle utiliza los bloqueos de tabla para impedir las inconsistencias debidas a la actividad LDD. Estos bloqueos evitan que, por ejemplo, un usuario elimine una tabla mientras otro usuario que tiene una transacción aún no comprometida está accediendo a dicha tabla. Oracle no utiliza ningún escalado de bloqueos para convertir los bloqueos de filas en bloqueos de tabla con el propósito de su control de concurrencia normal.

Oracle detecta los interbloqueos automáticamente y los resuelve haciendo que retroceda una de las transacciones involucradas en el interbloqueo.

Oracle soporta transacciones autónomas, que son transacciones independientes generadas por otras transacciones. Cuando Oracle invoca a una transacción autónoma genera una nueva transacción en un contexto separado. La nueva transacción se puede comprometer o retroceder antes de que el control vuelva a la transacción invocadora. Oracle soporta varios niveles de anidamiento de transacciones autónomas.

28.5.2. Estructuras básicas de recuperación

La tecnología flashback de Oracle, descrita en la Sección 28.5.1, se puede utilizar como mecanismo de recuperación, pero Oracle también admite la recuperación de medios en los que se hace una copia de seguridad física de los archivos. En esta sección se describen los mecanismos más tradicionales de copia de seguridad y recuperación.

Con el fin de comprender cómo se recupera Oracle de un fallo, como una caída del disco, es importante comprender las características básicas involucradas. Además de los archivos de datos que contienen las tablas e índices, existen archivos de control, registros históricos de rehacer, registros históricos de rehacer archivados y segmentos de retroceso.

El archivo de control contiene varios metadatos que son necesarios para operar en la base de datos, incluyendo la información sobre copias de seguridad.

Oracle registra cualquier modificación transaccional de una memoria intermedia de la base de datos en el registro histórico de rehacer, que consta de dos o más archivos. Registra la modificación como parte de la operación que la causa y sin considerar si la transacción finalmente se produce. Registra los cambios de los índices y segmentos de retroceso así como los cambios en la tabla de datos. Cuando se llenan los registros históricos rehacer se archivan mediante uno o varios procesos en segundo plano (si la base de datos se ejecuta en modo **archivelog**).

El segmento de retroceso contiene información sobre versiones anteriores de los datos (es decir, información para deshacer). Además de esta función, en el modelo de consistencia de Oracle la información se utiliza para restaurar la versión anterior de los datos cuando se deshace una transacción que ha modificado estos.

Para poder recuperar un fallo de almacenamiento se debería realizar periódicamente una copia de seguridad de los archivos de datos y de los archivos de control. La frecuencia de la copia de seguridad determina el tiempo de recuperación en el peor caso, puesto que la recuperación lleva más tiempo si la copia de seguridad es antigua. Oracle soporta copias de seguridad en caliente; copias de seguridad ejecutadas en una base de datos en línea que está sujeta a una actividad transaccional.

Durante la recuperación de una copia de seguridad, Oracle ejecuta dos pasos para alcanzar un estado consistente de la base de datos como existía antes del fallo. En primer lugar, Oracle rehace las transacciones aplicando los archivos históricos rehacer (archivados) a la copia de seguridad. Esta acción lleva a la base de datos al estado que existía en la fecha del fallo, pero no necesariamente a un estado consistente, puesto que los registros históricos de deshacer incluyen datos no comprometidos. En segundo lugar, Oracle deshace las transacciones no comprometidas mediante el uso del segmento de retroceso. La base de datos está ahora en un estado consistente.

La recuperación en una base de datos que ha sido objeto de una actividad transaccional grande debido a la última copia de seguridad puede ser costosa en tiempo. Oracle soporta la recuperación en paralelo, en la cual se utilizan varios procesos para aplicar información de rehacer simultáneamente. Oracle proporciona una herramienta GUI, el gestor de recuperación (*Recovery Manager*), que automatiza la mayor parte de las tareas asociadas con copias de seguridad y recuperación.

28.5.3. Oracle Data Guard

Para asegurar una alta disponibilidad, Oracle proporciona la característica base de datos en espera denominada **data guard** (esta característica es la misma que la de las copias de seguridad remotas, descrita en la Sección 16.9). Una base de datos en espera es una copia de la base de datos normal que se instala en un sistema separado. Si se produce un fallo catastrófico en el sistema principal, el sistema en espera se activa y asume el control, minimizando el efecto del fallo en la disponibilidad. Oracle mantiene la base de datos en espera actualizada mediante la aplicación constante de archivos históricos rehacer archivados que se envían desde la base de datos principal. La base de datos de seguridad se puede usar en línea en modo solo lectura y puede ser utilizada para informes y consultas para el apoyo a la toma de decisiones.

28.6. Arquitectura del sistema

Siempre que una aplicación de base de datos ejecuta una sentencia SQL hay un proceso del sistema operativo que ejecuta código en el servidor de la base de datos. Oracle se puede configurar de forma que el proceso del sistema operativo esté *dedicado* exclusivamente a la instrucción que se está procesando, o de forma que el proceso se pueda *compartir* entre varias sentencias. Esta última configuración, conocida como **servidor compartido,** tiene propiedades diferentes respecto a la arquitectura del proceso y la memoria. En primer lugar se estudiará la arquitectura del **servidor dedicado** y, posteriormente, la arquitectura del servidor multihebra.

28.6.1. Servidor dedicado: estructuras de memoria

La memoria utilizada por Oracle se divide principalmente en tres categorías: áreas de código software, que son las partes de la memoria en las que reside el código del servidor Oracle; área global del sistema (*system global area*: SGA) y área global del programa (*program global area*: PGA).

Para cada proceso, se asigna un PGA para albergar sus datos locales e información de control. Esta área contiene espacio en pilas para diversos datos de la sesión y la memoria privada para la instrucción SQL que se está ejecutando. También contiene memoria para operaciones de ordenación y asociación que pueden producirse durante la evaluación de la instrucción. El rendimiento de estas operaciones depende de la cantidad de memoria disponible. Por ejemplo, una reunión por asociación que se pueda realizar en memoria será más rápida que si es necesario acceder a disco. Dado que hay un gran número de operaciones de ordenación y asociación activas simultáneamente (ya que hay varias consultas y varias operaciones en cada una de ellas), la decisión sobre el tamaño de memoria a asignar para cada operación no es algo trivial, especialmente si fluctúa la carga de trabajo del sistema. Una baja asignación de memoria puede conllevar operaciones de E/S extra si una operación necesita acceder a disco, y una sobreasignación de memoria puede generar que se degrade el sistema. Oracle permite que el administrador de bases de datos especifique un parámetro *objetivo* para la cantidad total de memoria que se debería considerar disponible para estas operaciones. El tamaño de este objetivo normalmente estará basado en la cantidad total de memoria disponible en el sistema y algunos cálculos sobre cómo se debería dividir entre actividades de Oracle y no de Oracle. Oracle decidirá dinámicamente la mejor forma de dividir esta memoria entre las operaciones activas para maximizar la productividad. El algoritmo de asignación de memoria conoce la relación entre la memoria y el rendimiento de las diferentes operaciones y trata de asegurar que la memoria disponible se use de la forma más eficaz posible.

SGA es un área de memoria para estructuras que son compartidas entre los usuarios. Está formada por varias estructuras principales, incluyendo las siguientes:

- **Caché de memoria intermedia.** Esta caché mantiene bloques de datos a los que se accede frecuentemente (tablas e índices) en memoria para reducir la necesidad de ejecutar E/S a disco físico. Se usa la política «utilizado menos recientemente» salvo para los bloques a los que se acceda durante una exploración de tabla completa. Sin embargo, Oracle permite crear varias colas de memoria intermedia que tienen distintos criterios para la datación de los datos. Algunas operaciones Oracle omiten la caché de memoria intermedia y leen los datos directamente del disco.
- **Memoria intermedia de registro histórico de rehacer.** Esta memoria intermedia contiene la parte del registro histórico rehacer que no se ha escrito todavía en el disco.
- **Cola compartida.** Oracle busca maximizar el número de usuarios que pueden utilizar la base de datos concurrentemente minimizando la cantidad de memoria necesaria para cada usuario. Un concepto importante en este contexto es la capacidad

de compartir la representación interna de instrucciones SQL y el código procedimental escrito en PL/SQL. Cuando varios usuarios ejecutan la misma sentencia SQL pueden compartir la mayoría de estructuras de datos que representan el plan de ejecución de la instrucción. Solamente los datos que son locales para cada invocación específica de la instrucción necesitan mantenerse en una memoria privada.

Las partes que se pueden compartir de las estructuras de datos que representan la sentencia SQL se almacenan en la cola compartida, incluyendo el texto de la sentencia. El almacenamiento en caché de instrucciones SQL en la cola compartida también ahorra tiempo de compilación, puesto que una nueva invocación de la sentencia que ya está almacenada en caché no tiene que pasar por el proceso completo de compilación. La determinación de si una sentencia SQL es la misma que la existente en la cola compartida se basa en la coincidencia exacta del texto y en el establecimiento de ciertos parámetros de sesión. Oracle puede remplazar automáticamente las constantes en una sentencia SQL con variables vinculadas; las consultas futuras, que son iguales salvo por los valores de las constantes, coincidirán con la consulta anterior en la cola compartida.

La cola compartida también contiene cachés para información de diccionario y diversas estructuras de control. La caché de metadatos de diccionario es importante para mejorar el tiempo de compilación de las sentencias SQL. Además, la cola compartida se usa para las características de caché de resultados de Oracle.

28.6.2. Servidor dedicado: estructuras de proceso

Hay dos tipos de procesos que ejecutan código servidor Oracle: procesos servidor que procesan sentencias SQL y procesos en segundo plano que ejecutan diversas tareas administrativas relacionadas con el rendimiento. Algunos de estos procesos son opcionales y en ciertos casos se pueden utilizar varios procesos del mismo tipo por razones de rendimiento. Oracle puede generar unos doce tipos diferentes de procesos en segundo plano. Algunos de los tipos más importantes de procesos en segundo plano son:

- **Escritor de la base de datos.** Cuando se elimina una memoria intermedia de la caché de memoria intermedia, se debe volver a escribir en el disco si se ha modificado desde que se introdujo en la caché. Los procesos del escritor de la base de datos ejecutan esta tarea, lo que ayuda al rendimiento del sistema liberando espacio en la caché de la memoria intermedia.
- **Escritor del registro histórico.** El escritor del registro histórico procesa las entradas de escritura de la memoria intermedia del registro histórico rehacer al archivo del registro histórico rehacer en el disco. También escribe un registro de compromiso al disco siempre que se compromete una transacción.
- **Punto de revisión.** El proceso punto de revisión actualiza las cabeceras del archivo de datos cuando se produce un punto de revisión.
- **Monitor del sistema.** Este proceso lleva a cabo la recuperación ante una caída en caso necesario. También ejecuta cierta administración del espacio para reclamar espacio no utilizado en segmentos temporales.
- **Monitor de procesos.** Este proceso ejecuta la recuperación de procesos para procesos del servidor que fallan, liberando recursos y ejecutando diversas operaciones de limpieza.
- **Recuperador.** El proceso recuperador resuelve los fallos y dirige la limpieza de transacciones distribuidas.
- **Archivador.** El archivador copia el archivo de registro histórico de rehacer en línea a un registro histórico de rehacer cada vez que se llena el archivo de registro histórico en línea.

28.6.3. Servidor compartido

La configuración de servidor compartido aumenta el número de usuarios que un número dado de procesos servidor puede soportar compartiendo los procesos servidor entre las sentencias. Difiere de la arquitectura de servidor dedicado en los siguientes aspectos principales:

- Un proceso de envío en segundo plano encamina las solicitudes de los usuarios al siguiente proceso servidor disponible. Al realizar esto utiliza una cola de solicitudes y una cola de respuestas en el SGA. El distribuidor pone una nueva solicitud en la cola de solicitudes donde será recogida por un proceso servidor. Un proceso servidor completa una solicitud y pone el resultado en la cola de respuestas para ser recogida por el distribuidor y ser devuelta al usuario.
- Puesto que un proceso servidor se comparte entre varias instrucciones SQL, Oracle no mantiene datos privados en el PGA. Almacena los datos específicos de la sesión en el SGA.

28.6.4. Oracle Real Application Clusters

Las agrupaciones de aplicaciones reales de Oracle (*Oracle Real Application Clusters*: RAC) es una característica que permite que varios ejemplares de Oracle se ejecuten en la misma base de datos (recuerde que, en terminología de Oracle, un ejemplar es la combinación de procesos en segundo plano y áreas de memoria). Esta característica permite a Oracle ejecutarse en arquitecturas de hardware agrupadas y MPP (disco compartido y sin compartimiento). La capacidad de agrupar varios nodos tiene importantes ventajas en la dimensionabilidad y disponibilidad que son útiles en entornos OLTP y de almacén de datos.

Las ventajas de dimensionabilidad de la característica son obvias, puesto que más nodos implican más potencia de procesamiento. En las arquitecturas sin compartimiento, la adición de nodos a un agrupamiento normalmente requiere la redistribución de los datos entre los nodos.

Oracle usa una arquitectura de disco compartido en la que todos los nodos tienen acceso a todos los datos y, como resultado, se pueden añadir más nodos a una agrupación RAC sin preocuparse de que los datos se dividan entre los nodos. Oracle optimiza más todavía el uso del hardware a través de características tales como las reuniones por afinidad y por particiones.

RAC también se puede utilizar para lograr una alta disponibilidad. Si un nodo falla, los restantes todavía están disponibles para que la aplicación acceda a la base de datos. Las instancias restantes automáticamente retroceden las transacciones sin compromiso que están siendo procesadas en el nodo que falló con el fin de evitar un bloqueo de la actividad en el resto de nodos. RAC también permite el retroceso de parches, de forma que se pueden aplicar parches de software en un nodo cada vez sin necesidad de dejar fuera de servicio la base de datos.

La arquitectura de discos compartidos de Oracle evita muchos de los problemas de las arquitecturas sin compartición con los datos ya sean de discos locales a un nodo o no. Sin embargo el tener varios ejemplares ejecutándose con la misma base de datos da lugar a diversos problemas técnicos que no se dan con un único ejemplar.

Mientras que algunas veces es posible dividir una aplicación entre los nodos, de forma que los nodos raramente accedan a los mismos datos, siempre exsite la posibilidad de solapamiento, que afecta a la gestión de la caché.

Para conseguir una gestión eficiente de la caché en múltiples nodos la característica **mezcla de cachés** permite a los bloques de datos fluir directamente entre las cachés de distintos ejemplares mediante el uso de la interconexión, sin ser escritas a disco.

28.6.5. Administrador automático del almacenamiento

El administrador automático del almacenamiento (*Automatic Storage Manager*: ASM) es un administrador de volúmenes y de sistema de archivos desarrollado por Oracle. Aunque Oracle se puede utilizar con otros administradores de volúmenes y sistemas de archivos, así como dispositivos en crudo, ASM se ha diseñado específicamente para simplificar la administración del almacenamiento para las bases de datos de Oracle a la vez que se optimiza el rendimiento.

ASM administra colecciones de discos, denominados **grupos de discos,** y presenta una interfaz de sistema de archivos a la base de datos (recuerde que el espacio de tablas de Oracle se define en términos de archivos de datos). Ejemplos de qué puede constituir discos ASM son los discos o particiones de arrays de discos, los volúmenes lógicos y los archivos asociados en red. ASM automáticamente divide los datos entre los discos de un grupo y proporciona distintas opciones para los diversos niveles de espejo.

Si cambia la configuración de disco, por ejemplo, cuando se añaden más discos para aumentar la capacidad de almacenamiento, puede que haya que reequilibrar un grupo de discos de forma que los datos se distribuyan equitativamente entre todos ellos. La operación de equilibrado se puede realizar en segundo plano mientras la base de datos sigue totalmente operativa y con un mínimo impacto en el rendimiento de la misma.

28.6.6. Oracle Exadata

Exadata es un conjunto de bibliotecas de Oracle que se pueden ejecutar en CPU con array de almacenamiento sobre ciertos tipos de hardware de almacenamiento. Mientras que Oracle se basa fundamentalmente en una arquitectura de discos compartidos, Exadata aporta un matiz de no compartición en el hecho de que algunas operaciones que se ejecutarían normalmente en el servidor de la base de datos se trasladan a celdas de almacenamiento que solo acceden a datos locales para cada celda. (Cada celda de almacenamiento consta de un número de discos y varias CPU multinúcleo).

Se derivan ciertas ventajas de descargar ciertos tipos de procesamiento a CPU de almacenamiento:

- Permite una gran, pero relativamente barata, expansión de la cantidad de capacidad de procesamiento disponible.
- La cantidad de datos que se necesita transferir desde una celda de almacenamiento al servidor de base de datos puede reducirse drásticamente, lo que puede ser muy importante, ya que el ancho de banda entre la celda de almacenamiento y el servidor de la base de datos suele ser caro y, a menudo, un cuello de botella.

Cuando se ejecuta una consulta contra el almacenamiento Exadata, la reducción en la cantidad de datos que se necesita obtener proviene de varias técnicas que se trasladan a las celdas de almacenamiento y se ejecutan allí localmente:

- **Proyección.** Una tabla puede tener cientos de columnas, pero una consulta dada puede necesitar acceder solamente a un pequeño subconjunto de las mismas. Las celdas de almacenamiento pueden proyectar las columnas no necesarias y enviar solo las relevantes de vuelta al servidor de la base de datos.
- **Filtrado de tabla.** El servidor de la base de datos puede enviar una lista de predicados que son locales para una tabla a celdas de almacenamiento y solo serán devueltas al servidor las filas que casen con esos predicados.
- **Filtrado de reunión.** El mecanismo de filtrado permite que los predicados que son *filtros Bloom* también permitan filtrar las filas que son condiciones de reunión.

En combinación, trasladando estas técnicas a las celdas de almacenamiento se puede mejorar el procesamiento de consultas en órdenes de magnitud. Requiere que las celdas de almacenamiento, junto con el envío de vuelta de forma regular de bloques inalterados de la base de datos al servidor, pueda enviar de vuelta una versión compacta en la que se hayan eliminado ciertas filas y columnas. Esta capacidad requiere que el software de almacenamiento entienda el formato de bloques y los tipos de datos de Oracle e incluya las rutinas de evaluación de predicados y expresiones de Oracle.

Además de proporcionar ventajas en el procesamiento de consultas, Exadata también permite acelerar las copias de seguridad incrementales realizando un seguimiento de los cambios al nivel de bloque y devolviendo solo los bloques que hayan cambiado. Así mismo, el trabajo de dar formato cuando se crea el nuevo espacio de una nueva tabla se traslada al almacenamiento Exadata.

El almacenamiento Exadata soporta todas las características normales de Oracle y es posible disponer de una base de datos que incluya almacenamiento tanto Exadata como no Exadata.

28.7. Réplica, distribución y datos externos

Oracle proporciona soporte para la réplica y las transacciones distribuidas con compromiso de dos fases.

28.7.1. Réplica

Oracle soporta varios tipos de réplica (consulte la Sección 19.2.1 para una introducción a la réplica). En una forma sencilla, los datos de un sitio maestro se duplican en otros sitios en forma de *vistas materializadas*. Una vista materializada no tiene por qué contener todos los datos maestros, puede excluir, por ejemplo, ciertas columnas de una tabla por razones de seguridad. Oracle soporta dos tipos de vistas materializadas: *solo de lectura* y *actualizable*. Una vista materializada actualizable se puede modificar y las modificaciones se propagan hasta la tabla maestra. Sin embargo, las vistas materializadas solo de lectura se pueden definir en términos de conjuntos de operaciones sobre tablas en el sitio maestro. Los cambios en los datos maestros se propagan a las réplicas mediante el mecanismo de refresco de las vistas materializadas.

Oracle también soporta varios sitios maestros para los mismos datos, donde todos los sitios maestros actúan como pares. Se puede actualizar una tabla duplicada en cualquiera de los sitios maestro y la actualización se propaga al resto de sitios. Las actualizaciones se pueden propagar de forma asíncrona o sincrónica.

Para la réplica asíncrona, la información de actualización es enviada por lotes al resto de sitios maestros y a continuación se aplica. Puesto que los mismos datos podrían estar sujetos a modificaciones en conflicto en sitios diferentes, se podría necesitar una resolución del conflicto basada en algunas reglas del negocio. Oracle proporciona una serie de métodos de resolución de conflictos incorporados y permite a los usuarios escribir el suyo propio si fuera necesario.

Con la réplica asíncrona, una actualización de un sitio maestro se propaga de forma inmediata al resto de sitios.

28.7.2. Bases de datos distribuidas

Oracle soporta consultas y transacciones sobre varias bases de datos en distintos sistemas. Con el uso de pasarelas, los sistemas remotos pueden incluir bases de datos que no sean de Oracle. Oracle tiene capacidades incorporadas para optimizar una consulta que incluya tablas en distintos sitios, recuperar los datos relevantes y devolver los resultados como si hubiera sido una consulta local normal. Oracle también soporta la emisión transparente de transacciones a varios sitios mediante un protocolo incorporado de compromiso en dos fases.

28.7.3. Orígenes de datos externos

Oracle tiene varios mecanismos para soportar orígenes de datos externos. El uso más común es el almacén de datos cuando se cargan normalmente grandes cantidades de datos desde un sistema transaccional.

28.7.3.1. SQL*Loader

Oracle tiene una utilidad de carga directa, SQL*Loader, que soporta cargas rápidas en paralelo de grandes cantidades de datos desde archivos externos. Soporta una serie de formatos de datos y puede ejecutar varias operaciones de filtrado sobre los datos que se están cargando.

28.7.3.2. Tablas externas

Oracle permite hacer referencia a los orígenes de datos externos, tales como archivos planos, en la cláusula **from** de una consulta como si fueran tablas normales.

Una tabla externa se define mediante metadatos que describen los tipos de columna Oracle y la correspondencia entre los datos externos y dichas columnas. También es necesario un controlador de acceso para acceder a los datos externos. Oracle proporciona un controlador predeterminado para archivos planos.

La característica de tabla externa tiene el objetivo principal de operaciones de extracción, transformación y carga (ETL) en un entorno de almacén de datos. Los datos se pueden cargar en el almacén de datos desde un archivo plano utilizando:

create table *tabla* **as**
  select… **from**<tabla externa>
  where…

Mediante la agregación de operaciones sobre los datos en la lista **select** o la cláusula **where** se pueden realizar transformaciones y filtrados como parte de la misma sentencia SQL. Puesto que estas operaciones se pueden expresar en SQL nativo o en funciones escritas en PL/SQL o Java, la característica de tabla externa proporciona un mecanismo potente para expresar todas las clases de operaciones de transformación y filtrado de los datos. Para la dimensionabilidad se puede realizar en paralelo el acceso a la tabla externa con la ejecución en paralelo de Oracle.

28.7.3.3. Exportación e importación de datos en bruto

Oracle proporciona una utilidad de exportación para descargar datos y metadatos en archivos en bruto. Estos archivos son archivos normales que usan un formato propietario que se puede trasladar a otro sistema y cargarlos en otra base de datos de Oracle usando la correspondiente utilidad de importación.

28.8. Herramientas de gestión de bases de datos

Oracle proporciona a los usuarios una serie de herramientas y características para la gestión del sistema y el desarrollo de aplicaciones. En las últimas versiones de Oracle se ha puesto gran énfasis sobre el concepto de *administración*, es decir, en la reducción de la complejidad de todos los aspectos de la creación y administración de una base de datos Oracle. Este esfuerzo cubre una amplia variedad de áreas que incluyen la creación de bases de datos, el ajuste, la gestión del espacio y el almacenamiento, la copia de seguridad y la recuperación, la gestión de la memoria, los diagnósticos de rendimiento y la gestión de la carga de trabajo.

28.8.1. Oracle Enterprise Manager

El gestor corporativo de Oracle (*Oracle Enterprise Manager*: OEM) es la principal herramienta de Oracle para la gestión de sistemas de bases de datos. Proporciona una interfaz de usuario gráfica (GUI) sencilla de utilizar para la mayoría de las tareas asociadas con la administración de bases de datos Oracle, incluyendo la configuración, la supervisión del rendimiento, la gestión de los recursos, la seguridad y el acceso a los asesores. Además de la gestión de la base de datos, OEM proporciona la administración integrada de las aplicaciones de Oracle y un montón de software de bajo nivel.

28.8.2. Repositorio automático de carga de trabajo

El repositorio automático de carga de trabajo (*Automatic Workload Repository*: AWR) es uno de los componentes principales de la infraestructura proporcionada por Oracle para reducir el esfuerzo de la administración. Oracle supervisa la actividad en el sistema de bases de datos y registra diferentes tipos de información relativa a las cargas de trabajo y al consumo de recursos, registrándola en el AWR a intervalos regulares. Supervisando la carga de trabajo en el tiempo, Oracle puede detectar y ayudar en el diagnóstico de desviaciones de su comportamiento normal, como la degradación significativa en el rendimiento de alguna consulta, la contención de bloqueos o los cuellos de botella de la CPU.

La información recogida en el AWR proporciona la base para varios *asesores* que analizan diferentes aspectos del rendimiento y aconsejan cómo mejorarlo. Oracle tiene asesores para el ajuste de SQL, la creación de estructuras de acceso, como los índices y las vistas materializadas, y el dimensionamiento de la memoria. Oracle también proporciona asesores para la desfragmentación de segmentos y el redimensionamiento de deshacer.

28.8.3. Gestión de los recursos de la base de datos

Un administrador de la base de datos necesita poder controlar cómo se divide la potencia de procesamiento entre los usuarios y los grupos de usuarios. Algunos grupos pueden ejecutar consultas interactivas en las que el tiempo de respuesta es esencial; otros pueden ejecutar informes largos que se pueden ejecutar como tareas de procesos por lotes en segundo plano cuando la carga del sistema sea baja. También es importante poder evitar que un usuario envíe inadvertidamente una consulta ad hoc extremadamente costosa que retrasará demasiado al resto.

La característica de gestión de los recursos de la base de datos de Oracle permite al administrador de la base de datos dividir a los usuarios entre grupos consumidores de recursos con distintas prioridades y propiedades. Por ejemplo, un grupo de usuarios interactivos de alta prioridad pueden tener garantizado al menos un 60 por ciento de CPU. El resto, más alguna parte del 60 por ciento no utilizado por el grupo de alta prioridad, se asignaría entre los grupos de consumidores de recursos con menor prioridad. Un grupo de baja prioridad podría tener asignado un 0 por ciento, lo que significaría que las consultas enviadas por este grupo solo se ejecutarían cuando hubiera ciclos de CPU disponibles. Se pueden establecer para cada grupo límites para el grado de paralelismo necesario para la ejecución en paralelo.

El administrador de la base de datos también puede establecer el tiempo máximo de ejecución que se permite a una sentencia SQL. Cuando un usuario envía una sentencia, el gestor de recursos estima cuánto tiempo tardaría en ejecutarse y devuelve un error si la instrucción viola el límite. El gestor de recursos también puede limitar el número de sesiones de usuario que se pueden activar simultáneamente para cada grupo de consumidores de recursos. Otro recurso que puede controlar este gestor es el espacio para deshacer transacciones.

28.9. Minería de datos

El componente de minería de datos de Oracle (*Oracle Data Mining*) proporciona varios algoritmos que incorporan el proceso de minería de datos dentro de la propia base de datos tanto para la construcción de un modelo sobre un conjunto de datos de entrenamiento, como para su aplicación para evaluar los datos de producción reales. El hecho de que no sea necesario que los datos abandonen la base de datos es una ventaja significativa respecto a los motores separados de minería de datos. Tener que extraer e insertar grandes conjuntos de datos en un motor separado es incómodo y costoso, y además puede impedir que los nuevos datos se analicen justo después de que se introduzcan en la base de datos. Oracle proporciona algoritmos para el aprendizaje supervisado y sin supervisar, tales como:

- Clasificación: Bayes, modelos lineales generalizados, máquinas de soporte vectorial y árboles de decisión.
- Regresión: máquinas de soporte vectorial y modelos lineales generalizados.
- Importancia de los atributos: tamaño de descripción mínimo.
- Detección de anomalía: máquinas de soporte vectorial de una clase.
- Agrupamiento: agrupamiento de media-k mejorada y agrupamiento de división ortogonal.
- Reglas de asociación: a priori.
- Extracción de características: factorización de matrices no negativas.

Además, Oracle proporciona un gran número de funciones estadísticas incorporadas en la base de datos que cubren áreas como la regresión lineal, la correlación, la tabulación cruzada, el contraste de hipótesis, el ajuste de distribución y el análisis de Pareto.

Oracle proporciona dos interfaces para la minería de datos: una de Java y otra basada en el lenguaje procedimental PL/SQL de Oracle. Una vez que se construye un modelo en una base de datos de Oracle, se puede enviar o implantar en otras bases de datos de Oracle.

Notas bibliográficas

Se puede encontrar información actualizada, incluyendo documentación sobre productos Oracle, en los sitios web http://www.oracle.com y http://technet.oracle.com.

Los algoritmos inteligentes de Oracle para la gestión de memoria disponible para operaciones como la ordenación y la asociación se describen en Dageville y Zaït [2002]. Murthy y Banerjee [2003] describen XML Schemas. La compresión de tablas en Oracle se describe en Pöss y Potapov [2003]. El ajuste automático de SQL se describe en Dageville et ál. [2004]. El entorno del optimizador de transformación de consultas basadas en coste se describe en Ahmed et ál. [2006]. La característica de gestión de planes se trata en Ziauddin et ál. [2008]. Antoshenkov [1995] describe la técnica de compresión de mapas de bits alineadas por bytes utilizada en Oracle; consulte también Johnson [1999].

DB2 Universal Database de IBM

29

Sriram Padmanabhan

La familia de productos DB2 Universal Database, de IBM, es el buque insignia de los servidores de bases de datos y las familias de productos para inteligencia de negocio, integración de información y gestión de contenidos. El servidor de bases de datos DB2, Universal Database Server, está disponible en gran número de plataformas hardware y sistemas operativos. La lista de las plataformas del servidor soportadas incluye sistemas de alto nivel como *mainframes*, procesadores masivamente paralelos (MPP) y grandes servidores multiprocesadores simétricos (SMP); sistemas medios como SMP de cuatro y ocho vías; estaciones de trabajo, e incluso pequeños dispositivos de bolsillo. Los sistemas operativos soportados incluyen variantes de Unix tales como Linux, AIX de IBM, Solaris y HP-UX, así como Windows de Microsoft y MVS, VM, OS/400 de IBM, entre otros. DB2 Everyplace Edition soporta sistemas operativos tales como PalmOS y Windows CE. Existe incluso una versión gratis (sin cargo) de DB2, llamada DB2 Express-C. Las aplicaciones pueden migrarse sin problemas de plataformas de gama baja a grandes servidores gracias a la portabilidad de las interfaces y a los servicios de DB2. Además del motor del núcleo de la base de datos, la familia DB2 incluye otros productos que proporcionan herramientas, administración, replicación, acceso a datos distribuido, acceso ubicuo a datos, OLAP y muchas otras características. En la Figura 29.1 se describen los diferentes productos de la familia.

29.1. Visión general

El origen de DB2 se remonta al proyecto System R en el Centro de Investigación de Almadén (Almaden Research Center) de IBM (entonces denominado Laboratorio de Investigación de San José, IBM San Jose Research Laboratory). El primer producto DB2 se comercializó en 1984 sobre la plataforma *mainframe* de IBM, seguido tiempo después por versiones para otras plataformas. IBM ha mejorado continuamente DB2 en áreas tales como el procesamiento de transacciones (registro histórico de escritura anticipada y los algoritmos de recuperación ARIES), el procesamiento y optimización de consultas (Starburst), el procesamiento en paralelo (DB2 Parallel Edition), el soporte para bases de datos activas (restricciones y disparadores), las técnicas avanzadas de consultas y los almacenes de datos tales como vistas materializadas, agrupaciones multidimensionales, características «autónomas» y soporte del modelo relacional orientado a objetos (ADT, UDF).

Puesto que IBM soporta un gran número de plataformas de servidor y de sistemas operativos, el motor de base de datos DB2 dispone de cuatro bases de código diferentes: (1) Linux, Unix y Windows, (2) z/OS, (3) VM y (4) OS/400. Todos estos soportan un subconjunto común de lenguaje de definición de datos, de SQL y de las interfaces de administración. Sin embargo, los motores presentan características algo diferentes debido a los orígenes de cada plataforma. Este capítulo está centrado en el motor DB2 Universal Database (UDB) para Linux, Unix y Windows. Se reseñarán las características específicas de interés en otros sistemas DB2 cuando se considere apropiado.

La última versión de la base de datos universal (Universal Database, UDB) DB2 para Linux, Unix y Windows en 2009 es la versión 9.7. Esta versión contiene varias características nuevas, como el soporte para XML para entornos sin compartición, la compresión nativa de tablas e índices, la gestión automática del almacenamiento y soporte mejorado para lenguajes procedimentales como SQL PL y PL/SQL de Oracle.

- Servidores de bases de datos:
  - DB2 UDB para Linux, Unix, Windows.
  - DB2 UDB para z/OS.
  - DB2 UDB para OS/400.
  - DB2 UDB para VM/VSE.
- Inteligencia de negocio:
  - DB2 Data Warehouse Edition.
  - DB2 OLAP Server.
  - DB2 Alphablox.
  - DB2 CubeViews.
  - DB2 Intelligent Miner.
  - DB2 Query Patroller.
- Integración de datos:
  - DB2 Information Integrator.
  - DB2 Replication.
  - DB2 Connect.
  - Omnifind (para Enterprise Search).
- Gestión de contenidos:
  - DB2 Content Manager.
  - I BM Enterprise Content Manager.
- Desarrollo de aplicaciones:
  - IBM Rational Application Developer Studio.
  - DB2 Forms para z/OS.
  - QMF.
- Herramientas de gestión de bases de datos:
  - DB2 Control Center.
  - DB2 Admin Tool for z/OS.
  - DB2 Performance Expert.
  - DB2 Query Patroller.
  - DB2 Visual Explain.
- Bases de datos embebidas y móviles:
  - DB2e (Everyplace).

Figura 29.1. Familia de productos DB2.

29.2. Herramientas de diseño de bases de datos

La mayor parte de las herramientas de diseño de base de datos y herramientas CASE se pueden usar para diseñar una base de datos DB2. En particular, las herramientas de modelado de datos tales como ERWin y Rational Rose permiten al diseñador generar sintaxis LDD específica de DB2. Por ejemplo, la herramienta UML Data Modeler de Rational Rose puede generar instrucciones **create distinct type** del LDD específico de DB2 para tipos definidos por el usuario y usarlos posteriormente en definiciones de columnas. La mayor parte de las herramientas de diseño también soportan una característica de ingeniería inversa que lee las tablas del catálogo de DB2 y construye un diseño lógico para manipulaciones adicionales. Las herramientas pueden generar restricciones e índices.

DB2 proporciona constructores SQL para soportar muchas características lógicas y físicas de bases de datos usando SQL. Las características incluyen restricciones, disparadores y recursión usando construcciones de SQL. De igual forma, DB2 soporta ciertas características físicas de bases de datos tales como espacios de tablas, colas de memoria intermedia y particionamiento mediante el uso de sentencias SQL. La herramienta Centro de control de DB2 permite a los diseñadores o administradores emitir las instrucciones LDD apropiadas. Otra herramienta, *db2look*, permite al administrador obtener un conjunto completo de instrucciones LDD para una base de datos, incluyendo espacios de tablas, tablas, índices, restricciones, disparadores y funciones que crean una reproducción exacta del esquema de la base de datos para pruebas o réplica.

El Centro de control de DB2 incluye una serie de herramientas de diseño y administración. Para el diseño, el Centro de control proporciona una vista en árbol de un servidor, sus bases de datos, tablas, vistas y todos los demás objetos. También permite que los usuarios definan nuevos objetos, creen consultas SQL ad hoc y visualicen los resultados de las consultas. Las herramientas de diseño para ETL, OLAP, réplicas y federación también están integradas en el Centro de control. Toda la familia DB2 soporta el Centro de control para la definición de bases de datos, así como otras herramientas relacionadas. DB2 también proporciona módulos predeterminados para el desarrollo de aplicaciones en IBM Rational Application Development, así como en Microsoft Visual Studio.

29.3. Variaciones y extensiones de SQL

DB2 soporta un amplio conjunto de características de SQL para varios aspectos del procesamiento de bases de datos. Muchas de las características y la sintaxis de DB2 han proporcionado la base para los estándares SQL-92 o SQL-1999. En esta sección se resaltan las características XML del modelo relacional orientado a objetos y de integración de aplicaciones en la versión 8 de DB2 UDB, junto con algunas características de la versión 9.

29.3.1. Características XML

Se ha incluido en DB2 un extenso conjunto de funciones XML. A continuación se muestran algunas funciones XML importantes que se pueden utilizar en SQ, como parte de la extensión SQL/XML a SQL (descrita anteriormente en la Sección 23.6.3):

- **xmlelement.** Construye una etiqueta elemento con un nombre dado. Por ejemplo, **xmlelement**(*libro*) crea el elemento libro.
- **xmlattributes.** Construye el conjunto de atributos de un elemento.
- **xmlforest.** Construye una secuencia de elementos XML a partir de los argumentos.
- **xmlconcat.** Devuelve la concatenación de un número variable de argumentos XML.
- **xmlserialize.** Proporciona una versión del argumento serializada orientada a caracteres.
- **xmlagg.** Devuelve la concatenación de un conjunto de valores XML.
- **xml2clob.** Construye una representación del XML de tipo objeto de gran tamaño de caracteres **(clob).** Este **clob** puede ser recuperado por aplicaciones SQL.

Las funciones XML se pueden incorporar eficazmente en SQL para proporcionar una gran capacidad de manipulación de XML. Por ejemplo, suponga que se necesita construir un documento XML de pedido de compra de las tablas *pedidos*, *lineaproducto* y *producto* para el número de pedido 349. En la Figura 29.2 se muestra una consulta SQL con extensiones XML que se puede usar para crear dicho pedido de compra. La salida resultante se muestra en la Figura 29.3.

La versión 9 de DB2 soporta el almacenamiento nativo de datos XML como un tipo **xml** y soporte nativo para el lenguaje XQuery. Se han introducido técnicas de almacenamiento especializado, indexación, procesamiento de consultas y optimización para el procesamiento eficiente de datos XML y consultas en el lenguaje XQuery, además de API extendidas para tratar con los datos XML y XQuery.

29.3.2. Soporte para tipos de datos

DB2 proporciona soporte para tipos de datos definidos por el usuario (UDT, user-defined data types). Los usuarios pueden definir tipos de datos *distintos* o *estructurados*. Los tipos de datos distintos están basados en tipos de datos incorporados en DB2. Sin embargo, los usuarios pueden definir semánticas adicionales o alternativas para estos nuevos tipos. Por ejemplo, el usuario puede definir un tipo de datos distinto llamado *euro*, usando:

create distinct type *euro* **as decimal**(9,2);

Posteriormente, se puede crear un campo (por ejemplo, *precio*) en una tabla cuyo tipo sea *euro*. Este campo se puede usar en predicados de consultas como en el siguiente ejemplo:

select *producto* **from** *ventas_europa*
where *precio* > *euro*(1000);

```
select xmlemement(name 'PO',
   xmlattributes(idprod, fechapedido),
   (select xmlagg(xmlelement(name 'producto',
      xmlattributes(idproducto, cantidad, fechaenvío),
      (select xmlelement(name 'descproducto',
         xmlattributes(nombre, precio))
      from producto
      where producto.idproducto = líneaproducto.idproducto)))
   from líneaproducto
   where líneaproducto.idprod = pedidos.idprod))
from pedidos
where pedidos.idprod = 349;
```

Figura 29.2. Consulta XML en SQL de DB2.

```
<PO idprod = "349" fechapedido = "2004-10-01">
   <item idproducto = "1", cantidad = "10", fechaenvío = "2004-10-03">
   <itemdesc nombre = "IBM ThinkPad T41", precio = "1000.00 EUR"/>
   </item>
</PO>
```

Figura 29.3. Pedido de compra en XML para el producto 349.

Los tipos de datos estructurados son objetos complejos que normalmente están formados por dos o más atributos. Por ejemplo, el siguiente código declara un tipo de datos estructurado denominado *t_departamento*:

```
create type t_departamento as
  (nombredept varchar(32),
   directordept varchar(32),
   número integer)
mode db2/sql;

create type t_punto as
  (coord_x float,
   coord_y float)
mode db2/sql;
```

Los tipos estructurados se pueden usar para definir *tablas con tipos*:

```
create table departamento of t_departamento;
```

Es posible crear una jerarquía de tipos y tablas que puedan heredar métodos específicos y privilegios. Los tipos estructurados también sirven para definir atributos anidados dentro de una columna de una tabla. Aunque este tipo de definiciones violaría las reglas de normalización, pueden ser convenientes para aplicaciones orientadas a objetos que se basan en la encapsulación y en métodos bien definidos sobre los objetos.

29.3.3. Funciones y métodos definidos por el usuario

Otra característica importante es que los usuarios pueden definir sus propias funciones y métodos. Estas funciones se pueden incluir posteriormente en instrucciones y consultas SQL. Las funciones pueden generar escalares (único atributo) o tablas (fila multiatributo) como resultado. Los usuarios pueden definir funciones (escalares o de tablas) mediante el uso de la sentencia **create function.** Pueden escribir las funciones en lenguajes de programación comunes tales como C y Java, o lenguajes de guiones tales como REXX y PERL. Las funciones definidas por el usuario (FDU) pueden operar en los modos separado (*fenced*) y compartido (*unfenced*). En el modo separado las funciones se ejecutan mediante una hebra separada en su propio espacio de dirección. En el modo compartido se permite al agente de procesamiento de la base de datos ejecutar la función en el espacio de direcciones del servidor. Las FDU pueden definir un área de trabajo donde pueden mantener variables locales y estáticas en invocaciones diferentes. Por tanto, las FDU pueden realizar manipulaciones complejas de las filas intermedias que son su entrada. En la Figura 29.4 se muestra una definición de una FDU en DB2, *db2gse.GsegeFilterDist*, que apunta a un método externo específico que realiza realmente la operación.

Otra característica son los métodos asociados a un objeto, los cuales definen su comportamiento. A diferencia de las FDU, los métodos están asociados con tipos de datos estructurados particulares y se registran mediante el uso de la sentencia **create method.**

DB2 también soporta las extensiones procedurales para SQL, usando la extensión PL de SQL de DB2, incluyendo procedimientos, funciones y control de flujo. Las características procedurales de la norma SQL se describen en la Sección 5.2. Además, en la versión 9.7, DB2 también soporta la mayoría del lenguaje PL/SQL de Oracle, por compatibilidad con las aplicaciones desarrolladas para Oracle.

29.3.4. Objetos de gran tamaño

Las nuevas aplicaciones de las bases de datos requieren la manipulación de texto, imágenes, vídeo y otros tipos de datos típicos de gran tamaño. DB2 soporta estos requisitos proporcionando tres tipos de objetos de gran tamaño (*large object*: LOB) distintos. Cada LOB puede ocupar hasta 2 gigabytes. Los objetos de gran tamaño en DB2 son (1) objetos en binario (*binary large objetcs*: **blob**s), (2) objetos de caracteres de un único byte (*character large objects*: **clob**s) y (3) objetos de caracteres de dos bytes (*double byte character large objects*: **dbclob**s). DB2 organiza estos LOB como objetos separados, con cada fila en la tabla manteniendo punteros a sus LOB correspondientes. Los usuarios pueden registrar FDU que manipulen estos LOB según los requisitos de la aplicación.

29.3.5. Extensiones de índices y restricciones

Una característica reciente de DB2 permite a los usuarios crear extensiones de índices para generar claves a partir de datos estructurados usando la sentencia **create index extension.** Por ejemplo, se puede crear un índice en un atributo basado en el tipo de dato *t_departamento* definido anteriormente mediante la generación de claves con el nombre del departamento. El extensor espacial de DB2 utiliza el método de extensión de índices para crear índices como se muestra en la Figura 29.5.

Finalmente, los usuarios pueden aprovechar el rico conjunto de características de verificación de restricciones disponible en DB2 para forzar la semántica de los objetos tales como unicidad, validez y herencia.

```
create function db2gse.GsegeFilterDist (
    operacion integer, g1XMin double, g1XMax double,
    g1YMin double, g1YMax double, dist double,
    g2XMin double, g2XMax double, g2YMin double,
    g2YMax double )
  returns integer
  specific db2gse.GsegeFilterDist
  external name 'db2gsefn!gsegeFilterDist'
  language C
  parameter style db2 sql
  deterministic
  not fenced
  threadsafe
  called on null input
  no sql
  no external action
  no scratchpad
  no final call
  allow parallel
  no dbinfo;
```

Figura 29.4. Definición de una FDU.

```
create index extension db2gse. índice_espacial (
                              gS1 double, gS2 double, gS3 double)
from source key(geometry db2gse.ST_Geometry)
generate key using
  db2gse.GseGridIdxKeyGen(geometry..srid,
  geometry..xMin, geometry..xMax,
  geometry..yMin, geometry..yMax,
  gS1, gS2, gS3)
with target key(srsId integer,
  lvl integer, gX integer, gY integer, xMin double,
  xMax double, yMin double, yMax double)
search methods <condiciones> <acciones>
```

Figura 29.5. Extensión de índice espacial en DB2.

29.3.6. Servicios web

DB2 puede integrar servicios web como productor o consumidor. Se puede definir un servicio web para invocar DB2 usando sentencias de SQL. La llamada al servicio web resultante es procesada por un motor de servicios web incorporado en DB2, que genera la correspondiente respuesta SOAP. Por ejemplo, si hay un servicio web denominado *ObtenerActividadReciente*(*id_cliente*) que llame a la siguiente instrucción SQL, el resultado será la última transacción realizada para este cliente:

select *id_trn*, *importe*, *fecha*
from *transacciones*
where *id_cliente* = <input>
order by *fecha*
fetch first 1 **row only**;

La siguiente sentencia SQL muestra cómo actúa DB2 como un cliente de un servicio web. En este ejemplo, la función *ObtenerCotización*() definida por el usuario es un servicio web. DB2 realiza la llamada al servicio web mediante un motor incorporado de servicios web. En este caso, *ObtenerCotización* devuelve un valor de cotización para cada *id_acción* que aparece en la tabla *cartera*:

select *id_acción*, *ObtenerCotización*(*id_acción*)
from *cartera*;

29.3.7. Otras características

DB2 también soporta el producto de IBM Websphere MQ mediante la definición de las FDU apropiadas, tanto para interfaces de lectura como de escritura. Estas FDU pueden incluirse en sentencias de SQL para la lectura o escritura sobre colas de mensajes.

Desde la versión 9, DB2 soporta la autorización de grano fino mediante la función de control de acceso basado en etiquetas, que juega un papel similar al Virtual Private Database de Oracle (descrito anteriormente en la Sección 9.7.5).

29.4. Almacenamiento e indexación

La arquitectura de almacenamiento e indexación de DB2 está formada por la capa de sistema de ficheros o gestión de disco, los servicios para gestionar las memorias intermedias, objetos de datos tales como tablas, LOB, objetos índices y gestores de concurrencia y recuperación. Esta sección muestra una visión general de la arquitectura de almacenamiento. Además, en la siguiente sección se describe una nueva característica de la versión 8 de DB2 denominada agrupación multidimensional.

29.4.1. Arquitectura de almacenamiento

DB2 proporciona abstracciones de almacenamiento para gestionar tablas de bases de datos lógicas útiles en entornos multinodo (paralelo) y multidisco. En un sistema multinodo se pueden definir *grupos de nodos* para soportar la división de la tabla en conjuntos especificados de nodos. Esto aporta flexibilidad al asignar particiones de tabla a nodos diferentes de un sistema. Por ejemplo, las tablas de gran tamaño se pueden dividir entre todos los nodos de un sistema, mientras que las tablas pequeñas pueden residir en un único nodo.

Dentro de un nodo, DB2 usa *espacios de tablas* para organizar estas. Un espacio de tablas consta de uno o más *contenedores* que son referencias a directorios, dispositivos o archivos. Un espacio de tablas puede contener cero o más objetos de base de datos tales como tablas, índices o LOB. La Figura 29.6 ilustra estos conceptos. En esta figura se definen dos espacios de tablas para un grupo de nodos. Al espacio de tablas *rechumanos* se le asignan cuatro contenedores, mientras que al espacio de tablas *plan* solo se le asigna uno. Las tablas *empleado* y *departamento* se encuentran en el espacio de tablas *rechumanos*, mientras que la tabla *proyecto* está en el espacio de tablas *plan*. La distribución de datos asigna fragmentos (extensiones) de las tablas *empleado* y *departamento* a los contenedores del espacio de tablas *rechumanos*. DB2 permite al administrador crear tanto espacios de tablas gestionados por el *sistema* como por el *SGBD*. Los espacios de tablas gestionados por el sistema (*system-managed spaces*: SMS) son directorios o sistemas de archivo que mantienen el sistema operativo subyacente. En un SMS, DB2 crea objetos archivo en los directorios y asigna datos a cada uno de los archivos. Los espacios de tablas gestionados por el SGBD (*data managed spaces*: DMS) son dispositivos en bruto o archivos preasignados que son controlados por DB2. El tamaño de estos contenedores nunca puede crecer ni disminuir. DB2 crea mapas de asignación y gestiona el espacio de tablas DMS. En ambos casos la unidad de espacio de almacenamiento es una extensión de páginas. El administrador puede elegir el tamaño de la extensión para un espacio de tabla.

DB2 soporta la distribución en distintos contenedores como comportamiento predeterminado. Por ejemplo, cuando se insertan los datos en una tabla recientemente creada, la primera extensión se asigna a un contenedor. Una vez que la extensión está llena asigna los siguientes datos al siguiente contenedor por turnos rotatorios. La distribución proporciona dos ventajas significativas: E/S paralela y equilibrio de carga.

29.4.2. Colas de memorias intermedias

Se puede asociar una o varias memorias intermedias con cada espacio de tablas para gestionar diferentes objetos tales como datos e índices. Una memoria intermedia es un área de datos compartida que mantiene copias de objetos en la memoria. Estos objetos habitualmente están organizados en páginas para gestionar la memoria intermedia. DB2 permite la definición de memorias intermedias usando sentencias de SQL. La versión 8 de DB2 incluye la posibilidad de aumentar o disminuir el tamaño de las memorias intermedias de forma interactiva, o bien automáticamente si se selecciona la opción **automatic** como parámetro de configuración de memorias intermedias. Un administrador puede añadir más páginas a una memoria intermedia o bien disminuir su tamaño sin detener la actividad de la base de datos:

create bufferpool < cola-mem-intermedia > ...
alter bufferpool < cola-mem-intermedia > **size** <n>

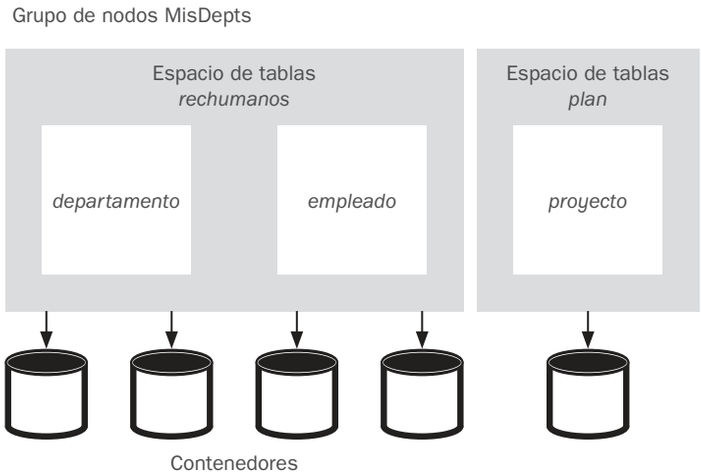


Figura 29.6. Espacios de tablas y contenedores en DB2.

DB2 también soporta la preextracción y las escrituras asíncronas mediante el uso de hebras separadas. El componente de gestión de datos de DB2 desencadena la preextracción de páginas de datos y de índices según los patrones de acceso de las consultas. Por ejemplo, una exploración de una tabla siempre desencadena la preextracción de páginas de datos. La exploración del índice puede desencadenar la preextracción de páginas de índices así como las páginas de datos, si se está accediendo de una forma agrupada. El número de preextracciones concurrentes, así como el tamaño de la preextracción, son parámetros configurables que es necesario establecer según el número de discos o contenedores en el espacio de tablas.

29.4.3. Tablas, registros e índices

DB2 organiza los datos relacionales como registros en las páginas. La Figura 29.7 muestra la vista lógica de una tabla y un índice asociado. La tabla consta de un conjunto de páginas. Cada página consta de un conjunto de registros (tanto registros de datos del usuario como registros especiales del sistema). La página cero de la tabla contiene registros del sistema especiales sobre la tabla y su estado. DB2 usa un registro del mapa de espacio denominado registro de control de espacio libre (*free space control record*: FSCR) para encontrar el espacio libre en la tabla. El registro FSCR normalmente contiene un mapa de espacio de 500 páginas. Una entrada FSCR consta de unos pocos bits que proporcionan una indicación aproximada del porcentaje de espacio libre en la página. El algoritmo de inserción o actualización debe validar las entradas del FSCR realizando una comprobación física del espacio disponible en la página.

Los índices también se organizan como páginas que contienen registros índice y punteros a páginas hijas y hermanas. DB2 proporciona soporte para los mecanismos de índices de árbol B^{+}. El índice de árbol B^{+} contiene páginas internas y páginas hoja. Los índices contienen punteros bidireccionales en el nivel hoja para soportar exploraciones hacia delante y hacia atrás. Las páginas hoja contienen entradas de índice que apuntan a los registros de la tabla. Cada registro de una tabla se puede identificar unívocamente usando su información de página y de ranura, que se denomina *identificador de registro* o RID (*record ID*).

DB2 admite «columnas incluidas» (include) en la definición del índice, como en:

create unique index I1 **on** T1 (C1) **include** (C2);

Las columnas del índice incluidas permiten a DB2 extender el uso de las técnicas de procesamiento «solo con índices» siempre que sea posible. Se pueden usar directivas adicionales tales como **minpctused** y **pctfree** para controlar la unión de páginas de índices y su asignación de espacio inicial.

La Figura 29.8 muestra el formato de datos típico en DB2. Cada página de datos contiene una cabecera y un directorio de ranuras. El directorio de ranuras es un array de 255 entradas que apuntan a los desplazamientos de los registros en la página. La figura muestra que el número de página 473 contiene el registro cero en el desplazamiento 3800 y el registro 2 en el desplazamiento 3400. La página 1056 contiene un registro 1 en el desplazamiento 3700, que es realmente un puntero hacia delante al registro <473,2>. Por ello, el registro <473,2> es un registro de desbordamiento que fue creado por una operación de actualización del registro <1056,1> original. DB2 soporta distintos tamaños de página tales como 4 KB, 8 KB, 16 KB y 32 KB. Sin embargo, cada página solo puede contener 255 registros de usuario. Los tamaños de página mayores son útiles en aplicaciones tales como almacén de datos, en las que la tabla contiene muchas columnas. Los tamaños de página menores son útiles para datos operacionales con frecuentes actualizaciones.

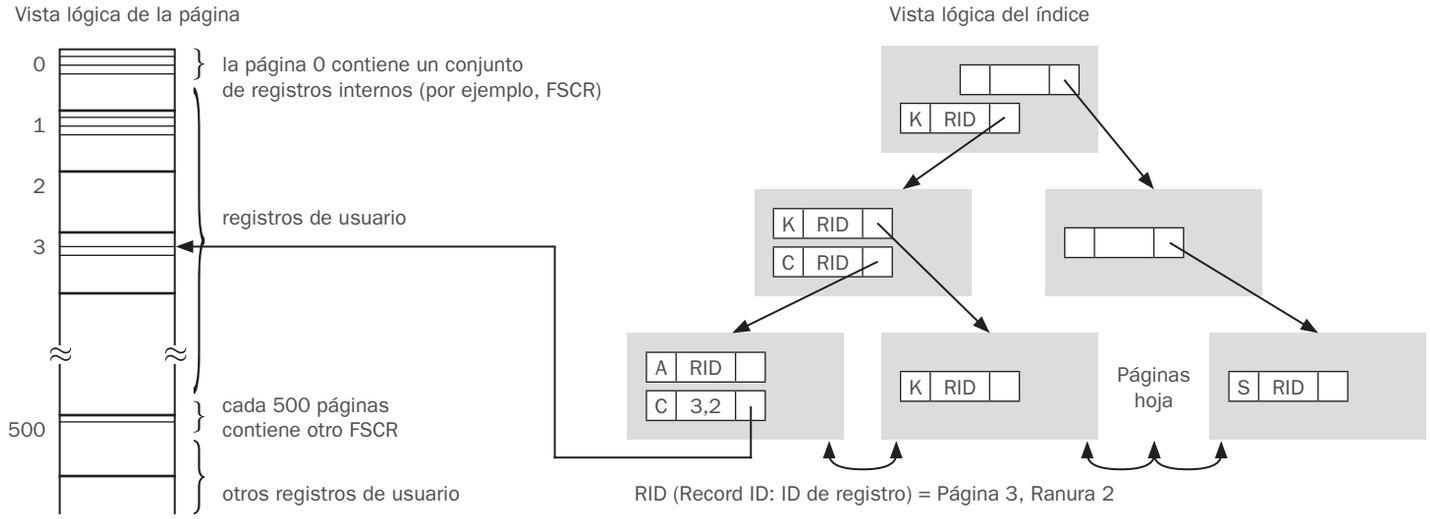


Figura 29.7. Vista lógica de tablas e índices en DB2.

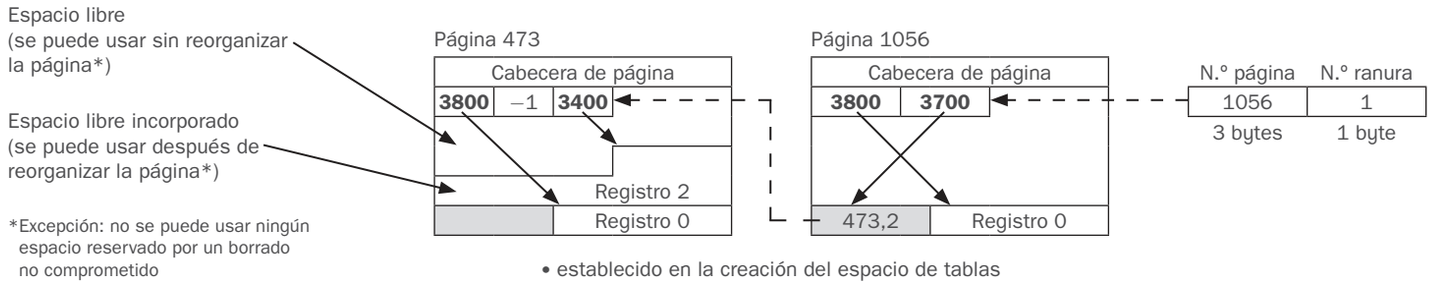


Figura 29.8. Diseño de las páginas de datos y de los registros en DB2.

29.5. Agrupación multidimensional

Esta sección proporciona una breve visión general de las principales características de las agrupaciones multidimensionales (*multidimensional clustering*: MDC). Con la agrupación multidimensional, una tabla DB2 puede crearse especificando una o varias claves como dimensiones para las que se agruparán los datos de la tabla. DB2 incluye una cláusula denominada **organize by dimensions** para este cometido. Por ejemplo, la siguiente LDD describe una tabla de ventas organizada tomando como dimensiones los atributos *idAlmacén*, *year*(*fechaPedido*) e *idProducto*:

create table *ventas*(*idAlmacén* **int**,
*fechaPedido***date**,
fechaEnvío **date**,
fechaRecepción **date**,
región **int**,
idProducto **int**,
precio **float**
añoPedido **int generated always as** *year*(*fechaPedido*))
organized by dimensions (*región*, *añoPedido*, *idProducto*);

Cada una de estas dimensiones puede estar formada por una o varias columnas, del mismo modo que los índices. De hecho, se crea automáticamente un «índice dimensional de bloques» (descrito más adelante) para cada una de las dimensiones especificadas, y se usa para acceder rápida y eficientemente a los datos. Si fuese necesario, se crea automáticamente un índice de bloque compuesto que contiene todas las columnas clave de la dimensión y que es utilizado para mantener la agrupación de datos durante las actividades de inserción y actualización.

Cada combinación única de valores dimensionales forma una «celda» lógica que está físicamente organizada como bloques de páginas, donde un bloque es un conjunto de páginas consecutivas en disco. El conjunto de los bloques que contienen páginas con datos que verifican un cierto valor de clave de uno de los índices de bloque de dimensión se denomina «banda». Cada página de la tabla forma parte de exactamente un bloque, y todos los bloques de la tabla están formados por el mismo número de páginas, que se denomina tamaño de bloque. DB2 asocia el tamaño de bloque con el tamaño de extensión del espacio de tablas, de forma que los límites de los bloques estén alineados con los límites de las extensiones.

La Figura 29.9 ilustra estos conceptos. Esta tabla MDC se encuentra agrupada por las dimensiones *year*(*fechaPedido*),[1] *región* e *idProducto*. La figura muestra un cubo lógico sencillo con solo dos valores por cada atributo de dimensión. En realidad, los atributos de dimensión pueden extenderse fácilmente a gran número de valores sin necesidad de administración. Las celdas lógicas están representadas en la figura por los cubos más pequeños. Los registros de la tabla se almacenan en bloques, que contienen una extensión de páginas consecutivas en disco. En el diagrama se representa cada bloque con una elipse gris, numerada de acuerdo al orden lógico de extensiones realizadas en la tabla. Solo se muestran algunos bloques de datos para la celda identificada por los valores de dimensión <1997, Canadá, 2>. Una columna o una fila de la retícula representan una banda para una dimensión particular. Por ejemplo, todos los registros que contienen el valor «Canadá» en la dimensión *región* se encuentran en los bloques contenidos en la banda definida por la columna «Canadá» del cubo. De hecho, todos los bloques de esta banda solo contienen registros que incluyen el término «Canadá» en el campo *región*.

1 Se pueden crear dimensiones utilizando una función generada.

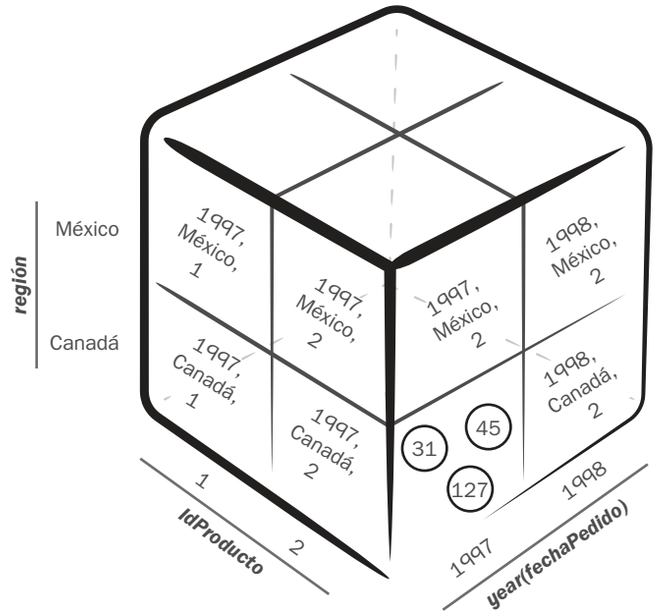


Figura 29.9. Vista lógica de la disposición física de una tabla MDC.

29.5.1. Índices de bloque

En el ejemplo anterior, se crea un índice dimensional de bloques sobre cada uno de los atributos región, *year*(*fechaPedido*) e *idProducto*. Todos los índices dimensionales de bloques se estructuran de la misma forma que un árbol B tradicional excepto que, en el nivel de las hojas, las claves apuntan a un *identificador de bloque* (*block identifier*: BID) en lugar de apuntar a un identificador de registro (*record identifier*: RID). Como cada bloque contiene potencialmente muchas páginas de registros, los índices de bloques son mucho más pequeños que los índices RID, y solo necesitan actualizarse cuando se añade un nuevo bloque a la celda, o cuando se vacían y eliminan bloques existentes en la celda. Las bandas, o el conjunto de bloques que contienen páginas con un valor clave particular en una dimensión, se representan en el índice dimensional de bloques asociadas mediante una lista de BID para ese valor clave. La Figura 29.10 ilustra las bandas de bloques para valores específicos de las dimensiones región e *idProducto*, respectivamente.

En el ejemplo anterior, para encontrar la banda que contiene todos los registros con valor «Canadá» en la dimensión *región* hay que buscar este valor clave en el índice dimensional de bloques y encontrar una clave como se muestra en la Figura 29.10a. Esta clave apunta al conjunto exacto de BID para ese valor particular.

Clave	Lista BID							
Canadá	21	31	45	77	127	376	501	719

(a) Entrada del índice dimensional de bloques para la *región* 'Canadá'.

Clave	Lista BID							
1	2	7	20	65	101	273	274	476

(b) Entrada del índice dimensional de bloques para el *idProducto* = 1

Figura 29.10. Entradas clave de índices de bloque.

29.5.2. Mapas de bloques

La tabla tiene también asociado un mapa de bloques. Este mapa registra el estado de los bloques de la tabla. Un bloque puede estar en diferentes estados tales como **in use, free, loaded** y **requiring constraint enforcement.** La capa de gestión de datos usa los estados del bloque para determinar diversas opciones de procesamiento. La Figura 29.11 muestra un ejemplo de mapa de bloques para una tabla.

El elemento 0 en el mapa de bloques representa el bloque 0 en el diagrama MDC de la tabla. El estado de disponibilidad del bloque es «U», que indica que está en uso. Sin embargo, es un bloque especial y no contiene registros de usuario. Los bloques 2, 3, 9, 10, 13, 14 y 17 no se están usando en la tabla, por lo que se consideran «F» (libres) en el mapa de bloques. Los bloques 7 y 18 se han cargado recientemente en la tabla. El bloque 12 fue cargado anteriormente y requiere la comprobación de restricciones.

0	1	2	3	4	5	6	7	8	9	19	11	12	13	14	15	16	17	18	19
U	U	F	F	U	U	U	L	U	F	F	U	C	F	F	U	U	F	L	...

Figura 29.11. Entradas del mapa de bloques.

29.5.3. Consideraciones de diseño

Un aspecto crucial de las MDC es la elección del conjunto correcto de dimensiones para agrupar una tabla y el tamaño correcto de bloque para minimizar el uso de espacio. Si las dimensiones y el tamaño de bloque se eligen apropiadamente, los beneficios de la agrupación se traducen en significativas ventajas de rendimiento y mantenimiento. Por otra parte, si se eligen incorrectamente, el rendimiento se puede degradar y el uso de espacio puede ser sensiblemente peor. Hay muchas técnicas de ajuste que se pueden explotar para organizar la tabla. Entre ellas se encuentran la variación del número de dimensiones, la granularidad de una o varias dimensiones, el tamaño de bloque (tamaño de extensión) y el tamaño de página del espacio de tablas que contiene la tabla. Se pueden usar una o varias de estas técnicas para identificar la mejor organización de la tabla.

29.5.4. Impacto sobre las técnicas existentes

Es natural preguntarse si la nueva característica de MDC tiene un impacto adverso o invalida algunas de las características existentes en DB2 para las tablas clásicas. Todas las características existentes, tales como índices RID secundarios, restricciones, disparadores, vistas materializadas definidas y opciones de procesamiento de consultas están disponibles para las tablas MDC. Por consiguiente, las tablas MDC se comportan exactamente como las tablas clásicas excepto por sus aspectos mejorados de agrupación y procesamiento.

29.6. Procesamiento y optimización de consultas

El compilador de consultas de DB2 transforma las consultas en un árbol de operaciones. DB2 usa entonces el árbol de operadores de la consulta en tiempo de ejecución para el procesamiento. DB2 soporta un rico conjunto de operadores de consulta que permiten considerar mejores estrategias de procesamiento y proporcionan flexibilidad en la ejecución de consultas complejas.

Las Figuras 29.12 y 29.13 muestran una consulta y su plan de consulta asociado. Se trata de una consulta compleja representativa (consulta 5) de la prueba TPC-H que contiene varias reuniones y agregaciones. El plan de consulta en este ejemplo es bastante simple, puesto que solamente se definen pocos índices y no están disponibles para esta consulta estructuras auxiliares como las vistas materializadas. DB2 proporciona varias características de «explicación» del plan, incluyendo una potente característica visual en el Centro de control que puede ayudar a los usuarios a comprender los detalles del plan de ejecución de la consulta. El plan de consulta en la figura está basado en la explicación visual de la consulta. La explicación visual permite al usuario comprender los costes y otras propiedades relevantes de las distintas operaciones de un plan de consulta.

DB2 transforma todas las consultas e instrucciones SQL, sin importar lo complejas que sean, en un árbol de consulta. La base o los operadores hoja del árbol de consulta manipulan los registros en las tablas de base de datos. Estas operaciones también se denominan *métodos de acceso*. Las operaciones intermedias del árbol incluyen operaciones del álgebra relacional tales como reuniones, operaciones de conjuntos y agregaciones. La raíz del árbol produce los resultados de la consulta o sentencia SQL.

```
– – ' Consulta TPCD de volumen de proveedor local (Q5)';
select nac_nombre, sum(lp_precioextendido *(1– lp_descuento))
                   as ingresos
from tpcd.cliente, tpcd.pedidos, tpcd.línea_pedido,
  tpcd.proveedor, tpcd.nación, tpcd.región
where cli_clave_cliente = ped_clave_cliente and
  ped_clave_pedido = lp_clave_pedido and
  lp_clave_proveedor = prv_clave_proveedor and
  cli_clave_nación = prv_clave_nación and
  prv_clave_nación = nac_clave_nación and
  nac_clave_región = reg_clave_región and
  reg_nombre = 'MIDDLE EAST' and
  ped_fecha_pedido >= date('1995-01-01') and
  ped_fecha_pedido < date('1995-01-01') + 1 year
group by nac_nombre
order by ingresos desc;
```

Figura 29.12. Consulta en SQL.

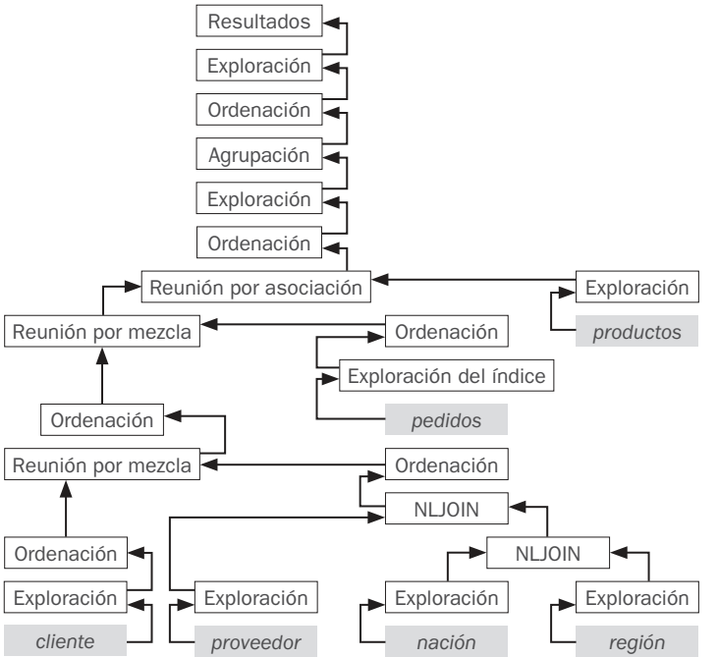


Figura 29.13. Plan de consulta en DB2 (explicación gráfica).

29.6.1. Métodos de acceso

DB2 soporta un conjunto detallado de métodos de acceso sobre tablas relacionales, incluyendo:

- **Exploración de tabla.** Con este método, el más básico, se accede a todos los registros de la tabla página por página.
- **Exploración de índice.** Se usa un índice para seleccionar los registros específicos que satisfacen la consulta. Accede a los registros usando los RID en el índice. DB2 detecta las posibilidades de la preextracción de las páginas de datos cuando observa un patrón de acceso secuencial.
- **Exploración de índice de bloques.** Es un nuevo método de acceso para tablas MDC. Se usa uno de los índices de bloque para explorar un conjunto específico de bloques de datos. DB2 accede y procesa los bloques seleccionados mediante operaciones de exploración de tablas de bloques.
- **Solo con el índice.** En este caso, el índice contiene todos los atributos que requiere la consulta. Por ello es suficiente una exploración de las entradas de índice. La técnica solo con el índice es normalmente una buena elección desde el punto de vista del rendimiento.
- **Lista de preextracción.** Este método de acceso es una buena elección para una exploración de índices no agrupada con un número significativo de RID. DB2 tiene una operación de ordenación de los RID y realiza una obtención de los registros en orden de las páginas de datos. El acceso ordenado cambia el patrón E/S de aleatorio a secuencial y también ofrece posibilidades de preextracción. La lista de preextracción se ha extendido para tratar también con índices de bloques.
- **Conjunción de índices de bloques y de registros.** DB2 usa este método cuando determina que se puede usar más de un índice para restringir el número de registros satisfactorios en una tabla base. Procesa el índice más selectivo para generar una lista de BID o RID. Después procesa el siguiente índice selectivo para devolver los BID o RID que encuentre. Un BID o RID solo requiere más procesamiento si está presente en la intersección (operación AND) de los resultados de la exploración del índice. El resultado de una operación AND del índice es una pequeña lista de RID o BID que se usan para extraer los registros correspondientes desde la tabla base.
- **Ordenación de índices de bloque y de registro.** Esta estrategia es una buena elección si se pueden usar dos o más bloques o índices de registros para satisfacer los predicados de la consulta que se combinan usando la operación OR. DB2 elimina los BID o RID duplicados realizando una ordenación y después extrae el conjunto de registros resultante. La disyunción de índices se ha extendido para considerar combinaciones de índices de bloque y RID.

Normalmente se envían todos los predicados de selección y proyección de una consulta a los métodos de acceso. Además, DB2 envía ciertas operaciones tales como la ordenación y la agregación, siempre que sea posible, con el fin de reducir el coste.

La característica de MDC aprovecha el nuevo conjunto de mejoras en los métodos de acceso para exploraciones de índices de bloques, preextracción de índices de bloques, conjunción de índices de bloques y disyunción de índices de bloques para procesar los bloques de datos.

29.6.2. Operaciones de reunión, de agregación y de conjuntos

DB2 soporta una serie de técnicas para estas operaciones. Para la reunión, DB2 puede elegir entre técnicas de bucles anidados, ordenación por mezcla y reunión por asociación. Para describir las operaciones binarias de reuniones y de conjuntos se usará la notación de las tablas «externas» e «internas» para distinguir los dos flujos de entrada. La técnica de bucles anidados es útil si la tabla interna es muy pequeña o se puede acceder usando un índice sobre un predicado de reunión. Las técnicas de reunión de ordenación por mezcla y reunión por asociación son útiles para reuniones que involucran tablas internas y externas grandes. Las operaciones de conjuntos se implementan mediante el uso de técnicas de ordenación y mezcla. La técnica de mezcla elimina los duplicados en el caso de la unión, mientras que los no duplicados se eliminan en el caso de la intersección. DB2 también soporta operaciones de reunión externa de todas las clases.

DB2 procesa las operaciones de agregación en modo impaciente o de envío siempre que sea posible. Por ejemplo, puede realizar la agregación mientras realiza la fase de ordenación. Los algoritmos de reunión y agregación aprovechan el procesamiento superescalar en CPU modernas usando técnicas orientadas a bloques y técnicas conscientes de la caché de memoria.

29.6.3. Soporte para el procesamiento de SQL complejo

Uno de los aspectos más importantes de DB2 es que usa la infraestructura de procesamiento de la consulta de forma extensible para soportar operaciones SQL complejas. Las operaciones SQL complejas incluyen soporte para subconsultas profundamente anidadas y correlacionadas, así como restricciones, integridad referencial y disparadores. Como la mayor parte de estas acciones están incluidas dentro del plan de consulta, DB2 está preparado para soportar una gran cantidad de restricciones y acciones. Las restricciones y comprobaciones de integridad se construyen como operaciones del árbol de consulta a partir de las instrucciones SQL de inserción, borrado o actualización. DB2 también soporta el mantenimiento de vistas materializadas mediante el uso de disparadores incorporados.

29.6.4. Procesamiento de consultas en multiprocesadores

DB2 extiende el conjunto base de operaciones de consulta con primitivas de intercambio de datos y control para soportar los modos SMP (es decir, memoria compartida), MPP (es decir, sin compartición) y SMP (es decir, discos compartidos) por agrupaciones del procesamiento de consultas. DB2 usa una abstracción «tabla-cola» para el intercambio de datos entre hebras sobre distintos nodos o sobre el mismo nodo. La tabla-cola es una memoria intermedia que redirige los datos a receptores apropiados mediante el uso de métodos de difusión, uno a uno o multidifusión dirigida. Las operaciones de control crean hebras y coordinan la operación de distintos procesos y hebras.

En todos estos modos DB2 usa un proceso coordinador para controlar las operaciones de colas y la reunión del resultado final. Los procesos de coordinación también pueden ejecutar algunas acciones globales de procesamiento de la base de datos si fuese necesario. Un ejemplo es la operación de agregación global para combinar los resultados de agregación local. Los subagentes o hebras esclavos ejecutan las operaciones base en uno o más nodos. En el modo SMP, los subagentes usan memoria compartida para sincronizarse entre sí cuando comparten datos. En un MPP, los mecanismos de tabla-cola proporcionan memoria intermedia y control de flujo para la sincronización entre distintos nodos durante la ejecución. DB2 usa técnicas exhaustivas de optimización y procesa consultas de forma eficiente en entornos MPP o SMP. La Figura 29.14 muestra una consulta simple ejecutándose en un sistema MPP de cuatro nodos. En este ejemplo, la tabla *ventas* está dividida entre los cuatro nodos $P_1 \ldots, P_4$. La consulta se ejecuta produciendo agentes que se ejecutan en cada uno de los nodos para explorar y filtrar las filas de la tabla *ventas* en ese nodo (denominado envío de funciones), y las filas resultantes se envían al nodo coordinador.

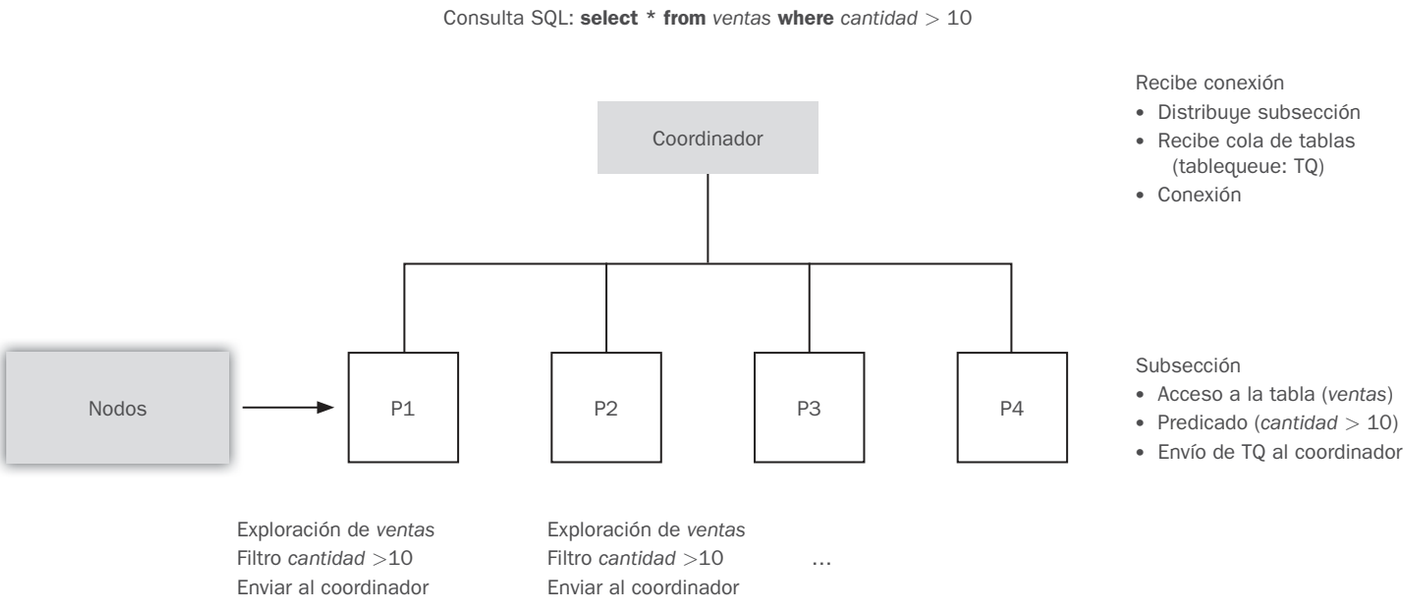


Figura 29.14. Procesamiento de consultas DB2 MPP usando envío de funciones.

29.6.5. Optimización de consultas

El compilador de consultas de DB2 utiliza una representación interna de la consulta, denominada modelo de grafos de consultas (*query graph model*: QGM) con el fin de ejecutar transformaciones y optimizaciones. Después de analizar la instrucción SQL, DB2 ejecuta transformaciones semánticas sobre el QGM para hacer cumplir las restricciones, la integridad referencial y los disparadores. El resultado de estas transformaciones es un QGM mejorado. Seguidamente, DB2 intenta ejecutar transformaciones de *reescritura de la consulta* que se consideran beneficiosas en la mayoría de las consultas. Se activan las reglas de reescritura, si son aplicables, para ejecutar las transformaciones requeridas. Los ejemplos de transformaciones de reescritura incluyen (1) descorrelación de subconsultas correlacionadas, (2) transformación de subconsultas en reuniones donde sea posible, (3) trasladar las operaciones **group by** bajo las reuniones si es aplicable y (4) uso de vistas materializadas para fragmentos de la consulta original.

El optimizador de consultas usa QGM mejorado y transformado como su entrada para la optimización. El optimizador se basa en el coste y usa un entorno extensible, controlado por reglas. El optimizador se puede configurar para operar a distintos niveles de complejidad. En el nivel más alto, usa un algoritmo de programación dinámica para considerar todas las opciones del plan de consulta y elige el plan de coste óptimo. En el nivel intermedio, el optimizador no considera determinados planes o métodos de acceso (por ejemplo, disyunción de índices) ni ciertas reglas de reescritura. En el nivel inferior de complejidad, el optimizador usa una heurística ávida simple para elegir un buen, aunque no necesariamente óptimo, plan de consulta. El optimizador emplea modelos detallados de las operaciones de procesamiento de la consulta (teniendo en cuenta detalles tales como el tamaño de la memoria y la preextracción) para obtener estimaciones adecuadas de los costes de E/S y CPU. Depende de la estadística de los datos para estimar la cardinalidad y selectividad de las operaciones. DB2 permite a un usuario generar histogramas de distribuciones en el nivel de las columnas y combinaciones de columnas mediante el uso de la utilidad **runstats.** Los histogramas contienen información sobre las apariciones del valor más frecuente así como sobre las distribuciones de frecuencia basadas en los cuartiles de los atributos. El optimizador de consultas utiliza estas estadísticas y genera el que considera el mejor plan de consulta interno y lo convierte en hebras de operadores y estructuras de datos asociados de la consulta para su ejecución mediante el motor de procesamiento de consultas.

29.7. Tablas de consultas materializadas

Las vistas materializadas en DB2 están permitidas en Linux, Unix y Windows, así como en las plataformas z/OS. Cualquier definición de vista sobre una o varias tablas o vistas puede ser una vista materializada. La utilidad de este tipo de vistas estriba en que mantienen una copia persistente de los datos de la vista para proporcionar un procesamiento de consultas más rápido. En DB2 estas vistas materializadas se denominan **tablas de consultas materializadas** (*materializad query tables*: MQT). Las MQT se especifican usando una instrucción **create table** como la mostrada en el ejemplo de la Figura 29.15.

En DB2 las MQT pueden referirse a otras MQT para crear un árbol o un bosque de vistas dependientes. Las MQT son muy ampliables, ya que pueden ser divididas en un entorno MPP y pueden tener claves de agrupación MDC. Las MQT resultan de máximo valor si el motor de la base de datos es capaz de encaminar perfectamente consultas hacia la MQT, y también cuando el motor de la base de datos puede mantenerlas eficientemente siempre que sea posible. DB2 proporciona ambas características.

```
create table empleado_departamento(id_departamento integer,
            id_empleado integer, nombre_empleado varchar(100),
            id_jefe integer) as
  select id_departamento, id_empleado, nombre_empleado, id_jefe
  from empleado, departamento
    data initially deferred
    refresh immediate – – (or deferred)
    maintained by user – – (or system)
```

Figura 29.15. Tablas de consultas materializadas de DB2.

29.7.1. Encaminamiento de consultas a MQT

La infraestructura del compilador de consultas de DB2 resulta ideal para impulsar toda la potencia de las MQT. El modelo interno del QGM permite al compilador comparar la consulta de entrada con las definiciones de MQT disponibles y elegir las MQT apropiadas. Después de la comparación, el compilador considera varias opciones de optimización. Entre ellas se encuentran tanto la consulta base como las versiones de reencaminamiento de MQT apropiadas. El optimizador itera a través de estas opciones antes de elegir la versión de ejecución óptima. El flujo completo de reencaminamiento y optimización se muestra en la Figura 29.16.

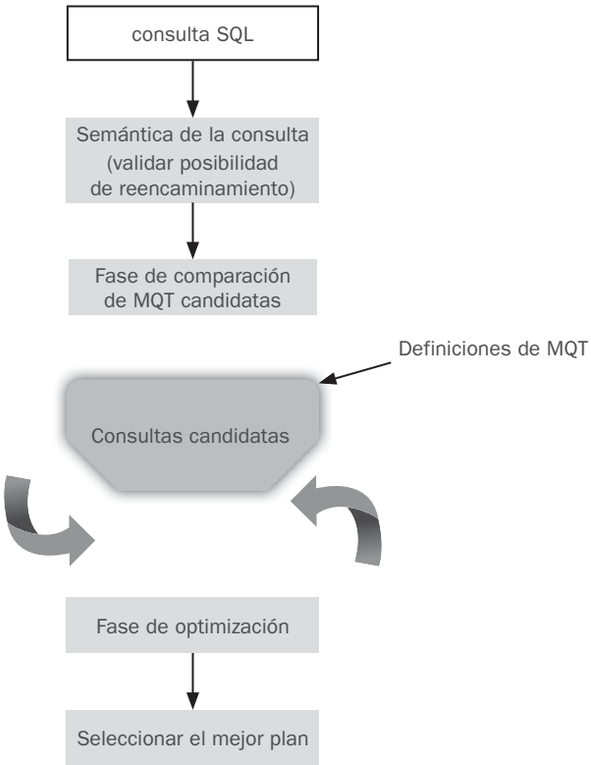


Figura 29.16. Comparación y optimización de MQT en DB2.

29.7.2. Mantenimiento de las MQT

Las MQT solo son útiles si el motor de la base de datos proporciona técnicas eficientes de mantenimiento. Hay dos dimensiones en el mantenimiento: tiempo y coste. En la dimensión temporal, las dos opciones son *inmediato* y *diferido*. DB2 soporta ambas. Si se selecciona mantenimiento inmediato, se crean disparadores internos que se compilan en las sentencias de **insert, update** o **delete** de los objetos fuente para procesar las actualizaciones de las MQT dependientes. En el caso del mantenimiento diferido, las tablas actualizadas se pasan a modo integridad y se debe ejecutar una instrucción **refresh** explícitamente para realizar la actualización. Con respecto a la dimensión de coste, se puede elegir entre *incremental* o *completo*. En el mantenimiento incremental solo se utilizan para el mantenimiento las filas que han sido actualizadas recientemente. El mantenimiento completo actualiza todas las MQT desde las fuentes. La matriz de la Figura 29.17 muestra ambas dimensiones y las opciones más útiles en cada una de ellas. Por ejemplo, el mantenimiento inmediato y el mantenimiento completo solo son compatibles si las fuentes son extremadamente pequeñas. DB2 también permite que las MQT sean mantenidas por el usuario **(user).** En este caso, la actualización de las MQT viene determinada por procesos explícitamente realizados por los usuarios usando SQL o utilidades.

Los siguientes comandos proporcionan un ejemplo sencillo de mantenimiento diferido de la vista materializada *emp_dept* después de una operación de carga de una de sus fuentes:

load from datosnuevos.txt **of type del**
insert into *empleado*;

refresh table *emp_dept*

Elecciones	*Incremental*	*Completa*
Inmediata	Sí, Después de insert/update/delete	Normalmente no
Diferida	Sí, Después de la carga	Sí

Figura 29.17. Opciones para el mantenimiento de MQT en DB2.

29.8. Características autónomas de B2

DB2 UDB proporciona varias características para simplificar el diseño y la administración de bases de datos. La informática autónoma engloba un conjunto de técnicas que permiten a un entorno informático administrarse a sí mismo y reducir dependencias externas frente a cambios en la seguridad externa e interna, carga del sistema u otros factores. La configuración, optimización, protección y supervisión son ejemplos de áreas que se benefician de las mejoras de la computación autónoma. En los siguientes apartados se describen las áreas de configuración y optimización.

29.8.1. Configuración

DB2 proporciona soporte de ajuste automático de diversos parámetros de configuración de memoria y del sistema. Por ejemplo, los parámetros de tamaño de la memoria intermedia y del montículo de ordenación se pueden especificar como automáticos. En este caso, DB2 supervisa el sistema y aumenta o disminuye lentamente estos tamaños en función de las características de la carga de trabajo.

29.8.2. Optimización

Las estructuras de datos auxiliares (índices, MQT) y las características de organización de datos (división, agrupación) son aspectos importantes para la mejora del rendimiento de las bases de datos en DB2. En el pasado, el administrador de la base de datos (DBA) tenía que basarse en su experiencia y en pautas conocidas para elegir índices, MQT, claves de división y claves de agrupación significativos. Dado el número potencial de opciones, ni siquiera los más expertos son capaces de encontrar la combinación exacta de estas características para una carga de trabajo dada en un tiempo limitado. DB2 introduce un asesor de diseño (*Design Advisor*) que proporciona sugerencias para todas estas características. El asesor de diseño analiza automáticamente la carga de trabajo y usa técnicas de optimización para presentar una serie de recomendaciones. La sintaxis del comando del asistente es:

```
db2advis -d <nombre BD> -i <archivo carga de trabajo> -m MICP
```

El parámetro «-m» permite que el usuario especifique las siguientes opciones:

- **M** — Tablas de consultas materializadas.
- **I** — Índices.
- **C** — Agrupaciones (MDC).
- **P** — Selección de claves de división.

El asesor utiliza todo el poder del marco de optimización de consultas de DB2 en estas recomendaciones. Usa una carga de trabajo y restricciones de tamaño y tiempo como parámetros de entrada. Al ser la base del marco de optimización de DB2, tiene un conocimiento completo del esquema y de las estadísticas de los datos subyacentes. El asesor emplea varias técnicas combinatorias para identificar índices, MQT, MDC y claves de división para mejorar el rendimiento de la carga de trabajo dada.

Otro aspecto de la optimización es el equilibrado de la carga de procesamiento en el sistema. En particular, las utilidades tienden a aumentar la carga en el sistema y provocan una reducción significativa del rendimiento en la carga de trabajo del usuario. Dada la tendencia hacia el empleo de utilidades en línea, hay una necesidad de equilibrar el consumo de la carga de las utilidades. DB2 introduce un mecanismo de regulación de carga. La técnica de regulación se basa en la teoría de control de realimentación. Ajusta y regula continuamente el funcionamiento de la utilidad de copia de seguridad usando parámetros específicos de control.

29.9. Herramientas y utilidades

DB2 proporciona una serie de herramientas para facilitar su uso y administración. Se ha aumentado y mejorado el núcleo de este conjunto de herramientas mediante un gran número de herramientas de otros fabricantes.

El Centro de control de DB2 es la herramienta principal para el uso y administración de bases de datos DB2. El Centro de control se ejecuta sobre muchas plataformas del tipo estación de trabajo. Está organizado a partir de objetos de datos tales como servidores, bases de datos, tablas e índices. Contiene interfaces orientadas a las tareas para ejecutar comandos y permite a los usuarios generar guiones SQL. La Figura 29.18 muestra una pantalla del panel principal del Centro de control y en ella se puede ver una lista de tablas en la base de datos *Sample* en la instancia DB2 sobre el nodo *Crankarm*. El administrador puede utilizar el menú para invocar un conjunto de herramientas componentes. Los componentes principales del Centro de control son: centro de comandos, centro de guiones, diario, centro de licencias, centro de alertas, supervisor del rendimiento, explicación visual, administración de bases de datos remotas, gestión de almacenamiento y soporte para la réplica. El centro de comandos permite a los usuarios y administradores ejecutar comandos de la base de datos y SQL. El centro de guiones permite a los usuarios ejecutar guiones SQL construidos de forma interactiva o desde un archivo. El supervisor del rendimiento permite al usuario supervisar varios eventos en el sistema de la base de datos y obtener instantáneas del rendimiento. «SmartGuides» proporciona ayuda para la configuración de parámetros del sistema DB2. Un constructor de procedimientos almacenados ayuda al usuario a desarrollar e instalar estos procedimientos. La explicación visual proporciona al usuario vistas gráficas del plan de ejecución de la consulta. El asistente de índices ayuda al administrador sugiriendo índices de rendimiento.

Aunque el Centro de control es una interfaz integrada de muchas de las tareas, DB2 también proporciona acceso directo a la mayoría de las herramientas. Para los usuarios, herramientas tales como el servicio de explicación, las tablas de explicación y la explicación gráfica proporcionan un análisis detallado de los planes de consulta. Los usuarios pueden usar el Centro de control para modificar las estadísticas (si tienen privilegios para ello) con el fin de generar los mejores planes de consulta.

Para los administradores, DB2 proporciona un soporte completo para la carga, importación, exportación, reorganización, redistribución y otras utilidades relacionadas con los datos. La mayor parte de las herramientas permiten su ejecución de forma incremental y en línea. Por ejemplo, se puede ejecutar un comando de carga en modo interactivo para permitir que las aplicaciones accedan a los contenidos originales de una tabla de forma concurrente. Las utilidades de DB2 están completamente preparadas para ejecutarse en modo paralelo.

Además, DB2 soporta varias herramientas tales como:

- Auditoría para el mantenimiento de la traza de auditoría de las acciones sobre la base de datos.
- Regulador para controlar la prioridad y los tiempos de ejecución en distintas aplicaciones.
- Supervisor de consultas para gestionar los trabajos de consulta en el sistema.
- Características de traza y diagnóstico para la depuración.
- Supervisión de eventos para seguir los recursos y eventos durante la ejecución del sistema.

DB2 para OS/390 tiene un gran conjunto de herramientas. QMF es una herramienta ampliamente utilizada para generar consultas ad hoc e integrarlas en aplicaciones.

29.10. Control de concurrencia y recuperación

DB2 soporta un completo conjunto de técnicas de control de concurrencia, aislamiento y recuperación.

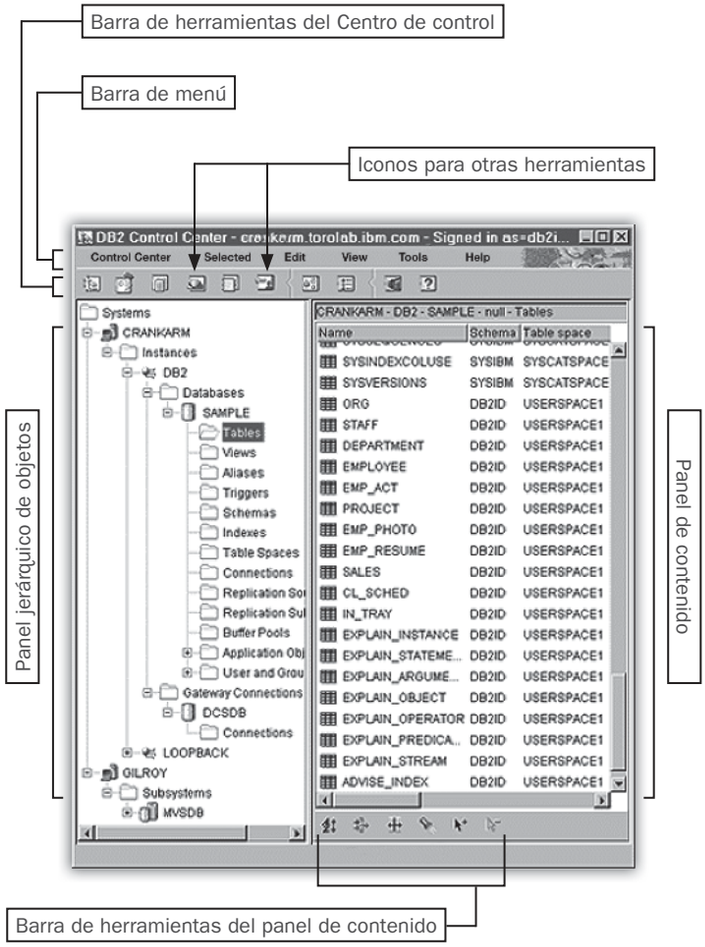


Figura 29.18. Centro de control de DB2.

29.10.1. Concurrencia y aislamiento

Para el aislamiento, DB2 soporta los modos *lectura repetible* (*repeatable read*: RR), *estabilidad en lectura* (*read stability*: RS), *estabilidad del cursor* (*cursor stability*: CS) y *lectura no comprometida* (*uncommitted read*: UR). Los modos RR, CS y UR no necesitan mayor explicación. El modo de aislamiento RS solo bloquea las filas que recuperan una aplicación en una unidad de trabajo. En una exploración posterior, la aplicación tiene garantizado ver todas estas filas (como RR), pero podría también ver nuevas filas. Sin embargo, esto podría ser un compromiso aceptable para algunas aplicaciones con respecto al aislamiento RR estricto. Normalmente, el nivel de aislamiento predeterminado es CS. Las aplicaciones pueden elegir el nivel de aislamiento en la fase de enlace. La mayoría de las aplicaciones comerciales disponibles soportan los distintos niveles de aislamiento y los usuarios pueden elegir la versión apropiada de la aplicación según sus necesidades.

Los distintos modos de aislamiento se implementan mediante el uso de bloqueos. DB2 soporta bloqueos en el nivel de registros y de tablas. Mantiene una estructura de datos de bloqueo de tablas separada del resto de la información de bloqueo. DB2 dimensiona el bloqueo del nivel de registros al de tablas si la tabla de bloqueos se llena, e implementa un bloqueo estricto de dos fases para todas las transacciones de actualización. Mantiene bloqueos de escritura y actualización hasta el momento del compromiso o retroceso. La Figura 29.19 muestra los distintos modos de bloqueo y sus descripciones. Los modos de bloqueo incluyen bloqueos intencionales en el nivel de tabla para maximizar la concurrencia. DB2 también implementa el bloqueo de clave siguiente y variaciones de este esquema para las actualizaciones que afecten a las exploraciones de índices con el fin de eliminar los problemas de lectura fantasma y de Halloween.

Modo de bloqueo	*Objetos*	*Interpretación*
IN (*intent none*: sin intención)	Espacios de tablas, tablas	Lectura sin bloqueos de filas
IS (*intent share*: intentar compartir)	Espacios de tablas, tablas	Lectura con bloqueos de filas
NS (*next key share*: siguiente clave compartido)	Filas	Bloqueos de lectura para los niveles de aislamiento RS o CS
S (*share*: compartido)	Filas, tablas	Bloqueo de lectura
IX (*intent exclusive*: intencional exclusivo)	Espacios de tablas, tablas	Intención de actualizar filas
SIX (*share with intent exclusive*: compartido intencional exclusivo)	Tablas	Sin bloqueos de lectura en las filas pero con bloqueos X en las filas actualizadas
U (*update*: actualización)	Filas, tablas	Bloqueo de actualización pero permitiendo leer a otros
NX (next-*key exclusive*: siguiente clave exclusivo)	Filas	Bloqueo de la siguiente clave para inserciones y borrados para prevenir las lecturas fantasma durante las exploraciones de índice RR
X (*exclusive*: exclusivo)	Filas, tablas	Solo se permiten lectores no comprometidos
Z (*superexclusive*: superexclusivo)	Espacios de tablas, tablas	Acceso completo exclusivo

Figura 29.19. Modos de bloqueo de DB2.

Una transacción puede seleccionar la granularidad de bloqueo en el nivel de tabla usando la sentencia **lock table.** Esto es útil para aplicaciones que saben que el nivel de aislamiento deseado es de nivel de tabla. Además, DB2 elige la granularidad de bloqueo apropiada para utilidades tales como reorg y load. Las versiones no interactivas de estas utilidades habitualmente bloquean la tabla en modo exclusivo, mientras que las versiones interactivas permiten a otras transacciones acceder concurrentemente realizando bloqueos de fila.

Por cada base de datos se activa un agente de detección de interbloqueos que comprueba periódicamente si se producen interbloqueos entre las transacciones. El intervalo de detección de interbloqueos es un parámetro configurable. Si se produce un interbloqueo, el agente elige una víctima y aborta su ejecución con un código de error de interbloqueo.

29.10.2. Compromiso y retroceso

Las aplicaciones pueden comprometerse o retroceder mediante el uso de las sentencias explícitas **commit** y **rollback.** Las aplicaciones también pueden utilizar sentencias **begin transaction** y **end transaction** para controlar el ámbito de las transacciones. No se soportan las transacciones anidadas. Normalmente, DB2 libera todos los bloqueos que se mantienen por una transacción en **commit** o **rollback.** Sin embargo, si se ha declarado una instrucción de cursor mediante la cláusula **withhold,** entonces se mantienen algunos bloqueos durante los compromisos.

29.10.3. Registro histórico y recuperación

DB2 implementa estrictamente el registro histórico y los esquemas de recuperación ARIES. Emplea el registro histórico de escritura anticipada para enviar registros del registro histórico al archivo de registro histórico persistente antes de que las páginas de datos se escriban o en el momento del **compromiso.** DB2 soporta dos tipos de modos de registro: registro histórico circular y registro de archivo. En el registro histórico circular, se utiliza un conjunto predefinido de archivos de registro histórico principal y secundario. El registro histórico circular es útil para la recuperación de caídas o la recuperación de un fallo de la aplicación. En el registro histórico de archivo, DB2 crea nuevos archivos de registro histórico y debe guardar los archivos de registro histórico antiguos con el fin de liberar espacio en el sistema de archivos. Los registros históricos de archivo son necesarios para la recuperación hacia adelante de una copia de seguridad de archivo. En ambos casos, DB2 permite al usuario configurar el número de archivos de registro histórico y los tamaños de los archivos de los registros históricos.

En entornos con muchas actualizaciones, DB2 puede configurarse para buscar compromisos en grupo con el fin de acumular las operaciones de escritura de archivo histórico.

DB2 soporta el retroceso de transacciones y la recuperación de caídas, así como recuperaciones por instantes (point-in-time) o hacia adelante (roll-forward). En el caso de la recuperación tras una caída, DB2 ejecuta las fases de *deshacer* estándar de procesamiento y procesamiento *rehacer* hasta y desde el último punto de revisión con el fin de recuperar el estado comprometido adecuado de la base de datos. Para la recuperación por instantes, se puede restaurar la base de datos desde una copia de seguridad y avanzar a un punto específico en el tiempo usando los archivos históricos guardados. El comando de recuperación hacia adelante soporta tanto los niveles de bases de datos como de espacios de tablas. También se pueden emitir en nodos específicos sobre un sistema multinodo. Recientemente se ha implementado un esquema de recuperación en paralelo para mejorar el rendimiento en sistemas multiprocesador SMP mediante el uso de muchas CPU. DB2 ejecuta la recuperación coordinada a través de nodos MPP mediante un esquema global de puntos de revisión.

29.11. Arquitectura del sistema

La Figura 29.20 de la página siguiente muestra algunos de los distintos procesos o hebras en un servidor DB2. Las aplicaciones cliente remotas se conectan al servidor de la base de datos empleando agentes de comunicación tales como *db2tcpcm*. Se asigna un agente a cada aplicación (agente coordinador en entornos MPP o SMP) denominado hebra *db2agent*. Este agente y sus agentes subordinados ejecutan las tareas relacionadas con la aplicación. Cada base de datos tiene un conjunto de procesos o hebras que ejecutan tareas tales como preextracción, limpieza de páginas de la cola de la memoria intermedia, archivo histórico y detección de interbloqueos. Finalmente, se encuentra disponible un conjunto de agentes en el entorno del servidor para ejecutar tareas tales como detección de caídas, servicios de licencia, creación de procesos y control de recursos del sistema. DB2 proporciona parámetros de configuración para controlar el número de hebras y procesos en un servidor. Casi todos los tipos distintos de agentes se pueden controlar mediante el uso de parámetros de configuración.

La Figura 29.21 muestra los distintos tipos de segmentos de memoria en DB2. La memoria privada en los agentes o hebras se utiliza principalmente para variables locales y estructuras de datos que solo son relevantes para la actividad actual. Por ejemplo, una ordenación privada podría asignar memoria desde el montículo privado del agente. La memoria compartida se divide en *memoria compartida del servidor*, *memoria compartida de la base de datos* y *memoria compartida de la aplicación*. El nivel de la base de datos de memoria compartida contiene estructuras de datos útiles tales como las colas de memoria intermedia, las listas de bloqueos, las cachés de los paquetes de aplicación y las áreas de ordenación compartida. Las áreas de memoria compartida del servidor y de la aplicación se usan principalmente para estructuras de datos y memorias intermedias de comunicaciones.

DB2 soporta varias colas de memoria intermedia para una base de datos. Las colas de memoria intermedia se pueden crear mediante el uso de la sentencia **create bufferpool** y se puede asociar con espacios de tablas. Es útil disponer de varias colas de memoria intermedia por diversas razones, pero se deberían definir después de un cuidadoso análisis de los requisitos de la carga de trabajo. DB2 soporta una completa lista de configuración de memoria y parámetros de ajuste. Esto incluye parámetros para todas las áreas de montículos de estructuras de datos grandes tales como las colas de memoria intermedia predeterminadas, el montículo de ordenación, la caché de paquetes, los montículos de control de la aplicación y el área de lista de bloqueos.

29.12. Réplicas, distribución y datos externos

La réplica de DB2 (*DB2 Replication*) es un producto de la familia DB2 que proporciona la réplica de datos entre DB2 y otros sistemas de bases de datos relacionales tales como Oracle, SQL Server de Microsoft, SQL Server de Sybase e Informix, y orígenes de datos no relacionales tales como IMS de IBM. Consiste en componentes *capturar* y *aplicar* que son controlados mediante interfaces de administración. Los mecanismos de captura de cambios pueden ser «basados en registros históricos» para tablas DB2, o «basados en disparadores» en el caso de otros orígenes de datos. Los cambios capturados se almacenan en áreas temporales de tablas bajo el control de DB2 Replication. Estas tablas intermedias con cambios se aplican después a las tablas destino mediante el uso de sentencias SQL normales: inserciones, actualizaciones y borrados. Las transformaciones basadas en SQL se pueden ejecutar sobre estas tablas intermedias usando condiciones de filtro, además de agregaciones. Las filas resultantes se pueden aplicar a una o más tablas destino. Los medios de administración controlan todas estas acciones.

DB2 soporta una característica denominada *réplica de colas* (*queue replication*). La réplica de colas (Q) crea un mecanismo de transporte de colas usando el producto de IBM de colas de mensajes para enviar registros de archivo históricos como mensajes. Estos mensajes se extraen de las colas en el receptor y se aplican a los destinos. El proceso de aplicación se puede paralelizar y tiene en cuenta las reglas de resolución de conflictos especificadas por el usuario.

Otro miembro de la familia DB2 es el producto integrador de información, que proporciona soporte para federación, réplica (usando el motor de réplica descrito anteriormente) y búsqueda. La edición federada integra tablas en bases de datos DB2 remotas u otras bases de datos relacionales en una única base de datos distribuida. Los usuarios y desarrolladores pueden acceder a diversas fuentes de datos no relacionales en formato tabular usando envolturas. El motor de la edición federada proporciona un método basado en el coste para la optimización de consultas entre los distintos sitios de datos.

DB2 soporta funciones de tabla definidas por el usuario que pueden permitir el acceso de orígenes de datos no relacionales y externos. Se pueden crear funciones de tabla definidas por el usuario mediante la sentencia **create function** con la cláusula **returns table.** Con estas características, DB2 puede participar en los protocolos OLE-DB.

Finalmente, DB2 proporciona soporte completo para el procesamiento de transacciones distribuidas mediante el protocolo de compromiso en dos fases. DB2 puede actuar como coordinador o agente para el soporte XA distribuido. Como coordinador, DB2 puede ejecutar todos los estados del protocolo de compromiso de dos fases. Como participante, DB2 puede interactuar con cualquiera de los administradores de transacciones distribuidas comerciales.

29.13. Características de inteligencia de negocio

DB2 Data Warehouse Edition es un producto de la familia DB2 que incorpora características de inteligencia de negocio. Esta edición tiene como base el motor de DB2, y lo mejora con características para ETL, OLAP, minería y generación interactiva de informes. El motor de DB2 proporciona dimensionabilidad mediante sus características MPP. En el modo MPP, DB2 puede soportar configuraciones dimensionables a varios cientos de nodos para bases de datos de gran tamaño (terabytes). Adicionalmente, las características como MDC y MQT proporcionan soporte para los requisitos de procesamiento de consultas complejas de la inteligencia de negocio.

Otro aspecto de la inteligencia de negocio es el procesamiento analítico interactivo (*on-line analytical processing*: OLAP). La familia DB2 incluye un componente denominado *vista de cubos* que proporciona un mecanismo para construir estructuras de datos apropiadas para MQT dentro de DB2 que se puedan usar para procesamiento OLAP relacional. La vista de cubos proporciona soporte para el modelado de cubos multidimensionales y un mecanismo de correspondencia con un esquema relacional en estrella. Este modelo se usa para recomendar los MQT, índices y definiciones MDC apropiados para mejorar el rendimiento de consultas OLAP sobre la base de datos. Además, las vistas de cubos pueden aprovechar el soporte nativo en DB2 para las operaciones **cube by** y **rollup** para la generación de cubos agregados. La vista de cubos es una herramienta que se puede usar para integrar estrechamente DB2 con distribuidores de productos OLAP como Business Objects, Microstrategy y Cognos.

Además, DB2 también proporciona soporte multidimensional OLAP usando el servidor OLAP de DB2. Este servidor puede crear un almacén de datos (*data mart*) desde una base de datos DB2 para realizar análisis usando técnicas OLAP. El servidor OLAP de DB2 usa el motor OLAP del producto Essbase.

DB2 Alphablox es una nueva característica que permite crear informes y realizar análisis interactivamente en línea. Un aspecto muy atractivo de Alphablox es la posibilidad de construir rápidamente formularios de análisis basados en web usando un enfoque fundamentado en bloques constructivos denominados *blox*.

Para análisis avanzados, DB2 Intelligent Miner proporciona diversos componentes para modelar, clasificar (*scoring*) y visualizar datos. La minería de datos permite a los usuarios realizar clasificaciones, predicciones, agrupaciones, segmentaciones y asociaciones en grandes conjuntos de datos.

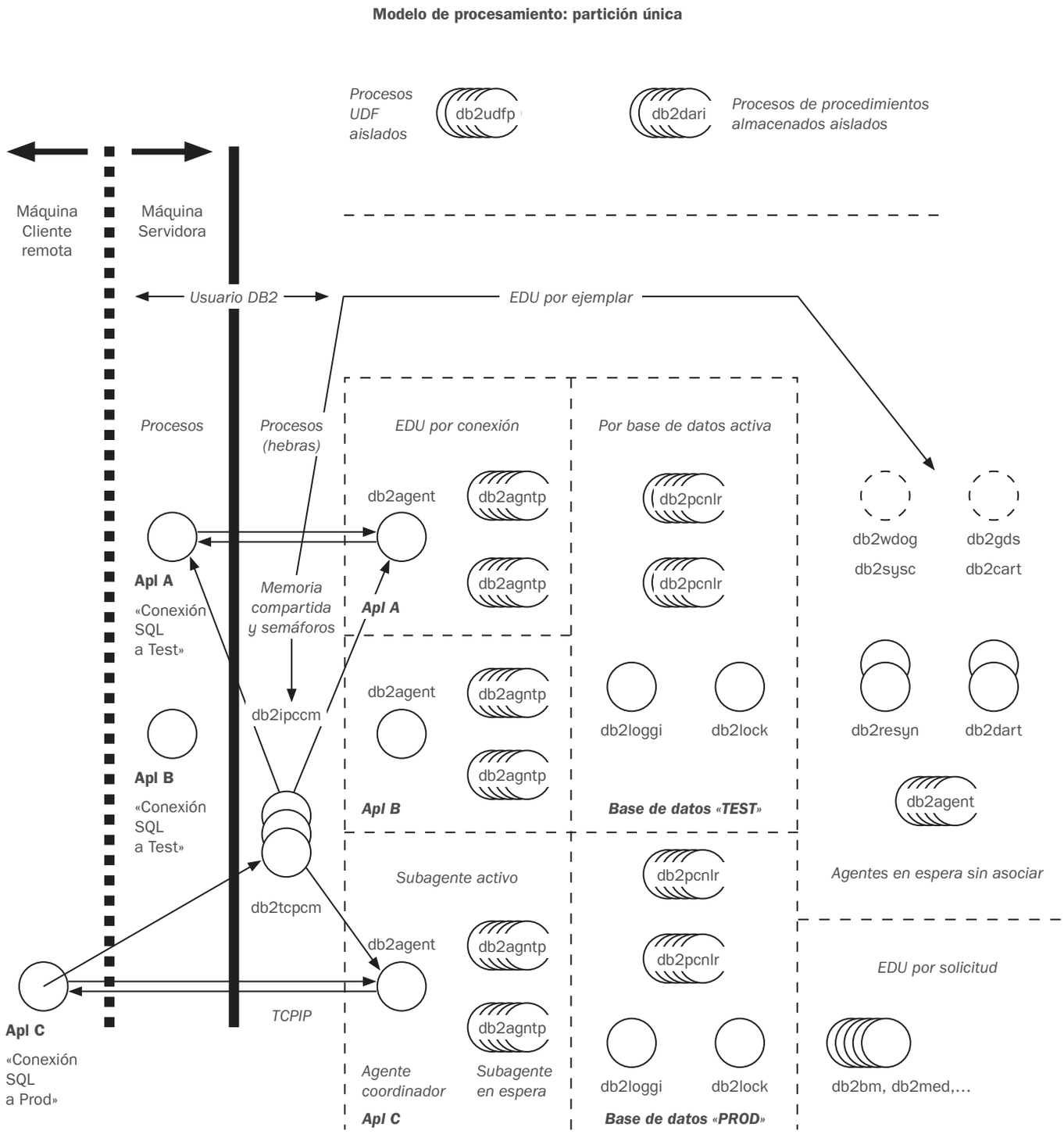


Figura 29.20. Modelo de procesos en DB2.

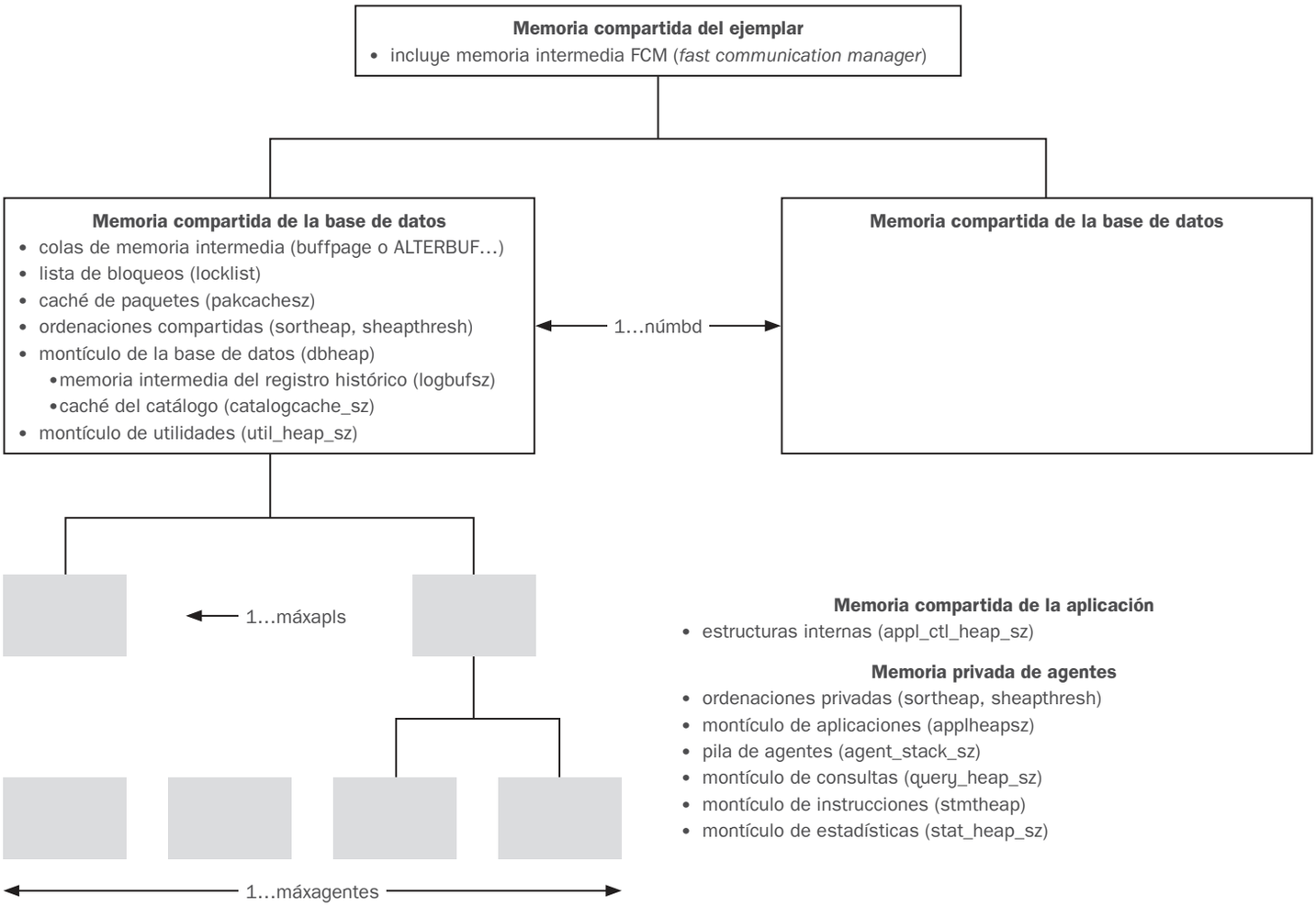


Figura 29.21. Modelo de memoria DB2.

Notas bibliográficas

El origen de DB2 se remonta al proyecto del Sistema R (Chamberlin et ál. [1981]). Las contribuciones de investigación de IBM incluyen áreas tales como el procesamiento de transacciones (archivo histórico con escritura anticipada y algoritmos de recuperación ARIES) (Mohan et ál. [1992]); el procesamiento y optimización de consultas (Starburst) (Haas et ál. [1990]); el procesamiento paralelo (DB2 Parallel Edition) (Baru et ál. [1995]); el soporte para bases de datos activas (restricciones, disparadores) (Cochrane et ál. [1996]), técnicas avanzadas de consultas y la gestión de almacenes de datos tales como vistas materializadas (Zaharioudakis et ál. [2000], Lehner et ál. [2000]), agrupaciones multidimensionales (Padmanabhan et ál. [2003], Bhattacharjee et ál. [2003]), características autónomas (Zilio et ál. [2004]) y soporte del modelo relacional orientado a objetos (ADT, UDF) (Carey et ál. [1999]). Se puede encontrar más información sobre el procesamiento de consultas en multiprocesadores en Baru et ál. [1995]. Los libros de Don Chamberlin proporcionan una buena revisión de SQL y las características de programación de versiones anteriores de DB2 (Chamberlin [1996], Chamberlin [1998]). Los primeros libros de C. J. Date y otros proporcionan un buen repaso de las características de DB2 para OS/390 (Date [1989], Martin et ál. [1989]).

Los manuales de DB2 ofrecen revisiones definitivas de cada versión de DB2. La mayoría de estos manuales están disponibles en línea (http://www.software.ibm.com/db2). Otros libros sobre DB2 para desarrolladores y administradores son Gunning [2008], Zikopoulos et ál. [2004], Zikopoulos et ál. [2007] y Zikopoulos et ál. [2009].

Microsoft SQL Server 30

Sameet Agarwal, José A. Blakeley, Thierry D'Hers, Ketan Duvedi, César A. Galindo-Legaria, Gerald Hinson, Dirk Myers, Vaqar Pirzada, Bill Ramos, Balaji Rathakrishnan, Jack Richins, Michael Rys, Florian Waas, Michael Zwilling

SQL Server de Microsoft es un sistema de gestión de bases de datos relacionales que se puede instalar tanto en computadoras portátiles y de sobremesa como en servidores corporativos, con una versión compatible, basada en el sistema operativo Windows Mobile, disponible para dispositivos móviles como Pokect PC, SmartPhones y Centros multimedia portátiles. SQL Server se desarrolló originalmente en los años ochenta en SyBase para sistemas UNIX y posteriormente Microsoft lo tradujo a sistemas Windows NT. Desde 1994, Microsoft ha distribuido versiones de SQL Server desarrolladas independientemente de Sybase, que dejó de usar el nombre SQL Server a finales de los años noventa. La última versión, SQL Server 2008, está disponible en las ediciones exprés (Express), estándar (Standard) y corporativa (Enterprise), y se ha traducido a muchos idiomas de todo el mundo. En este capítulo el término SQL Server se refiere a todas estas ediciones de SQL Server 2008.

SQL Server proporciona servicios de réplica entre varias copias de SQL Server y con otros sistemas de bases de datos. Su componente Analysis Services (servicios de análisis), una parte esencial del sistema, incluye servicios de procesamiento analítico en línea (OLAP, Online Analytical Processing) y de minería de datos. SQL Server proporciona una gran colección de herramientas gráficas y de «asistentes» que guían a los administradores de las bases de datos en tareas como la configuración de copias de seguridad periódicas, la duplicación de datos entre los distintos servidores y el ajuste del rendimiento de las bases de datos. Muchos entornos de desarrollo soportan SQL Server, incluidos Visual Studio de Microsoft y los productos relacionados, en especial los productos y servicios .NET.

30.1. Herramientas para la administración, el diseño y la consulta de las bases de datos

SQL Server proporciona un conjunto de herramientas para gestionar todos los aspectos del desarrollo, la consulta, el ajuste, la prueba y la administración de SQL Server. La mayor parte de estas herramientas giran alrededor de Management Studio, de SQL Server. Management Studio ofrece una interfaz de comandos común para la administración de todos los servicios asociados con SQL Server, que incluye el motor de la base de datos (Database Engine), los servicios de análisis (Analysis Services), los servicios de informes (Reporting Services), el servidor móvil de SQL Server (SQL Server Mobile) y los servicios de integración (Integration Services).

30.1.1. Desarrollo de bases de datos y herramientas visuales para las bases de datos

Al diseñar una base de datos, el administrador de la base de datos crea objetos de esta como las tablas, las columnas, las claves, los índices, las relaciones, las restricciones y las vistas. Management Studio de SQL Server proporciona acceso a herramientas visuales como ayuda para la creación de estos objetos en las bases de datos. Estas herramientas ofrecen tres mecanismos de ayuda para el diseño de las bases de datos: Database Designer (diseñador de bases de datos), Table Designer (diseñador de tablas) y View Designer (diseñador de vistas).

Database Designer es una herramienta visual que permite al propietario de la base de datos y a sus delegados crear tablas, columnas, claves, índices, relaciones y restricciones. En esta herramienta el usuario puede interactuar con los objetos de la base de datos mediante los diagramas de la base de datos, que muestran de forma gráfica la estructura de la misma. View Designer proporciona una herramienta visual de consulta que permite al usuario crear y modificar vistas de SQL mediante el uso de las posibilidades de arrastrar y soltar de Windows. La Figura 30.1 muestra una vista abierta desde Management Studio.

30.1.2. Herramientas para la consulta y el ajuste de las bases de datos

Management Studio de SQL Server proporciona varias herramientas de ayuda al proceso de desarrollo de las aplicaciones. Se pueden desarrollar y probar inicialmente las consultas y los procedimientos almacenados mediante Query Editor (editor de consultas), que sustituye al analizador de consultas de SQL Server. Query Editor soporta la creación y edición de guiones para varios entornos, incluyendo Transact-SQL, el servicio de guiones de SQL Server SQLCMD, el lenguaje de expresión multidimensional MDX que se usa para el análisis de datos, el lenguaje de minería de datos de SQL Server DMX, el lenguaje de análisis de XML XMLA y SQL Server Mobile. Se pueden realizar más análisis usando ServerProfiler de SQL. Las recomendaciones de ajuste de las bases de datos las proporciona una tercera herramienta: Database Tuning Advisor.

30.1.2.1. Query Editor

Query Editor proporciona una interfaz de usuario gráfica sencilla para la ejecución de consultas de SQL y el examen de sus resultados, así como una representación gráfica de **showplan,** los pasos elegidos por el optimizador para la ejecución de cada consulta. Query Editor está integrado con Object Explorer de Management Studio, que permite que el usuario arrastre objetos o tablas a las ventanas de las consultas y ayuda a crear sentencias **select, insert, update** y **delete** para cualquier tabla.

El administrador de la base de datos puede usar Query Editor para:

- **Analizar consultas:** Query Editor puede mostrar el plan de ejecución de cualquier consulta en formato gráfico o de texto, así como presentar las estadísticas relativas al tiempo y a los recursos necesarios para su ejecución.
- **Dar formato a las consultas de SQL:** incluidos el sangrado y la codificación sintáctica por colores.
- **Usar plantillas para los procedimientos almacenados, las funciones y las sentencias básicas de SQL:** Management Studio incluye docenas de plantillas predefinidas para la creación de comandos de LDD, y los usuarios también pueden definir las suyas propias.

La Figura 30.2 muestra Management Studio y Query Editor con el plan gráfico de ejecución de una consulta que supone una reunión de cuatro tablas y una agregación.

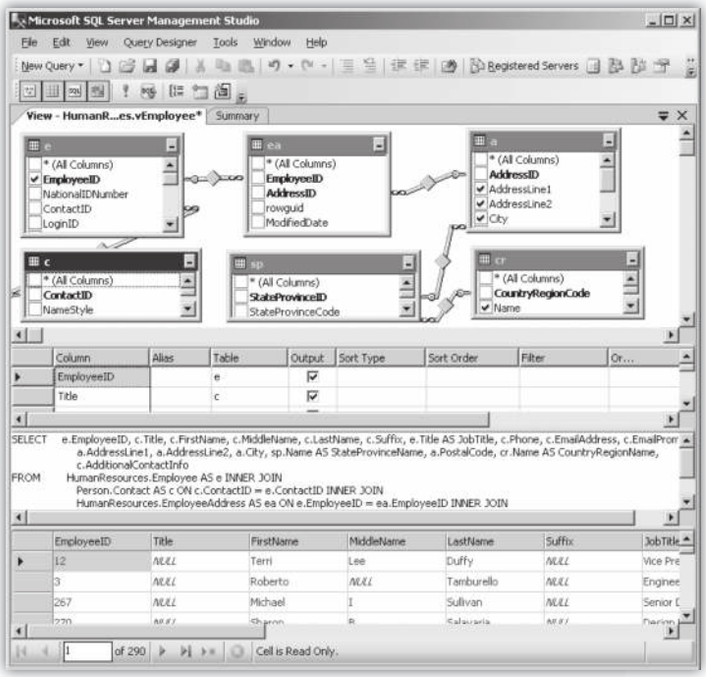


Figura 30.1. View Designer abierto para la vista HumanResources.vEmployee.

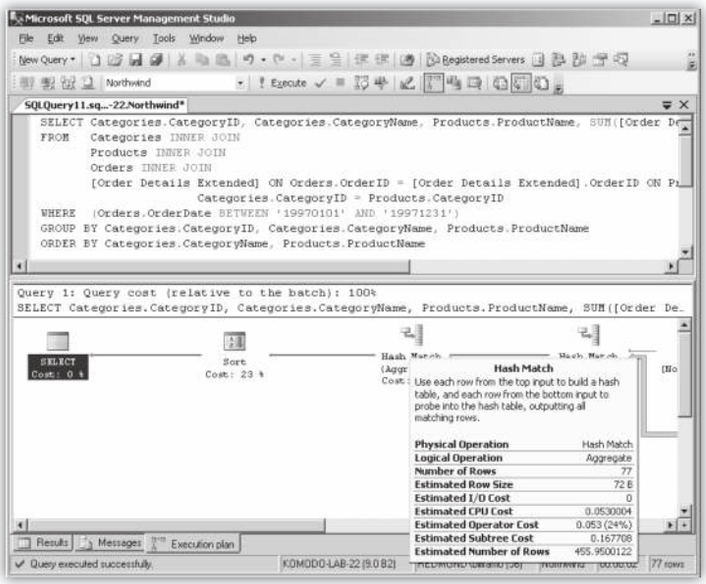


Figura 30.2. Plan de ejecución de una reunión de cuatro tablas con la agregación **group by.**

30.1.2.2. SQL Profiler

SQL Profiler es una utilidad gráfica que permite a los administradores de bases de datos supervisar y registrar la actividad de Database Engine y de Analysis Services de SQL Server. Puede mostrar toda la actividad del servidor en tiempo real o crear filtros que se centren en acciones de usuarios, aplicaciones o tipos de comandos concretos. También permite mostrar cualquier sentencia o procedimiento almacenado de SQL enviado a cualquier ejemplar de SQL Server (si los privilegios de seguridad lo permiten), así como los datos de rendimiento que indican el tiempo que la consulta ha tardado en ejecutarse, la cantidad de CPU y de E/S necesarias y el plan de ejecución utilizado.

SQL Profiler permite ahondar aún más en SQL Server para supervisar automáticamente cada sentencia ejecutada como parte de un procedimiento almacenado, cada operación de modificación de datos, cada bloqueo adquirido o liberado, o cada ocasión en que crece un archivo de la base de datos. Se pueden capturar docenas de eventos distintos y, para cada evento, se pueden capturar docenas de elementos de datos. SQL Server divide realmente la funcionalidad de seguimiento en dos componentes diferentes, aunque conectados. SQL Profiler es el servicio de traza del lado cliente. Mediante SQL Profiler los usuarios pueden decidir guardar los datos capturados en un archivo o en una tabla, además de mostrarlos en la interfaz de usuario (UI) de Profiler. Esta herramienta muestra todos los eventos que cumplen el criterio del filtro en el momento en que se producen. Una vez que se han guardado los datos de la traza, SQL Profiler puede leer los datos guardados para mostrarlos o analizarlos.

En el lado del servidor está el servicio de traza de SQL, que gestiona las colas de los eventos generados por los productores de eventos. Una hebra (subproceso en DB2) consumidora lee los eventos desde las colas y los filtra antes de enviarlos al proceso que los ha solicitado. Los eventos son la unidad principal de actividad en lo que se refiere a la traza, y un evento puede ser cualquier cosa que suceda dentro de SQL Server o entre SQL Server y un cliente. Por ejemplo, la creación o eliminación de un objeto, la ejecución de un procedimiento almacenado, la adquisición o liberación de un bloqueo y el envío de un archivo de procesamiento por lotes de Transact-SQL desde un cliente a SQL Server son eventos. Hay un conjunto de procedimientos almacenados del sistema para definir los eventos que hay que seguir, los datos de cada evento que son interesantes y el lugar en el que se debe guardar la información recogida de los eventos. Los filtros aplicados a los eventos pueden reducir la cantidad de información recogida y almacenada.

SQL Server garantiza que se recoja siempre cierta información esencial, y se puede usar como un mecanismo útil de auditoría. SQL Server está certificado para el nivel C2 de seguridad y muchos de los eventos de los que se puede realizar una traza solo están disponibles para cumplir los requisitos de la certificación C2.

30.1.2.3. Database Tuning Advisor

Las consultas y las actualizaciones se pueden ejecutar mucho más rápido si se dispone de un conjunto de índices adecuado. El diseño de los mejores índices posibles para las tablas de una base de datos de gran tamaño es una tarea compleja: no solo exige un conocimiento completo del modo en que SQL Server usa los índices y de la manera en que el optimizador de consultas toma las decisiones, sino también del modo en que las aplicaciones y las consultas interactivas usan realmente los datos. Database Tuning Advisor (DTA) de SQL Server es una potente herramienta para el diseño de los mejores índices y de las mejores vistas indexadas (materializadas) posibles, de acuerdo con las cargas de trabajo de consultas y de actualizaciones observadas.

DTA puede ajustar varias bases de datos a la vez y basa sus recomendaciones en una carga de trabajo que puede ser un archivo de eventos de traza capturados, un archivo con instrucciones de SQL o un archivo de datos de XML. SQL Profiler está diseñado para capturar todas las instrucciones de SQL remitidas por todos los usuarios en un periodo de tiempo determinado. DTA puede examinar posteriormente los patrones de acceso a los datos de todos los usuarios, las aplicaciones y las tablas, y realizar recomendaciones equilibradas.

30.1.3. Management Studio de SQL Server

Además de proporcionar acceso a las herramientas de diseño de bases de datos y a las herramientas visuales para bases de datos, Management Studio de SQL Server, que es fácil de usar, soporta la administración centralizada de todos los aspectos de varias instalaciones de Database Engine, Analysis Services, Reporting Services, Integration Services y SQL Server Mobile, incluidos la seguridad, los eventos, las alertas, la programación, las copias de seguridad, la configuración del servidor, los ajustes, la búsqueda de texto completo y la réplica. Management Studio de SQL Server permite que el administrador de la base de datos cree, modifique y copie los esquemas y los objetos de las base de datos de SQL Server como las tablas, las vistas y los disparadores. Como varias instalaciones de SQL Server se pueden organizar en grupos y ser tratadas como una unidad, Management Studio de SQL Server puede gestionar cientos de servidores de manera simultánea.

Aunque se puede ejecutar en la misma computadora que el motor de SQL Server, Management Studio de SQL Server ofrece las mismas posibilidades de gestión cuando se ejecuta en cualquier máquina de Windows 2000 (o posterior). Además, la arquitectura eficiente cliente-servidor de SQL Server hace práctico el uso de las posibilidades de acceso remoto (acceso telefónico a redes) de Windows para la administración y la gestión.

Management Studio de SQL Server libera al administrador de la base de datos de tener que conocer los pasos y la sintaxis concretos necesarios para completar cada trabajo. Ofrece asistentes que guían a los administradores de bases de datos en el proceso de configuración y mantenimiento de las instalaciones de SQL Server. La interfaz de Management Studio se muestra en la Figura 30.3 e ilustra la manera en que se puede crear de forma directa un guion para las copias de seguridad de las bases de datos a partir de los cuadros de diálogo.

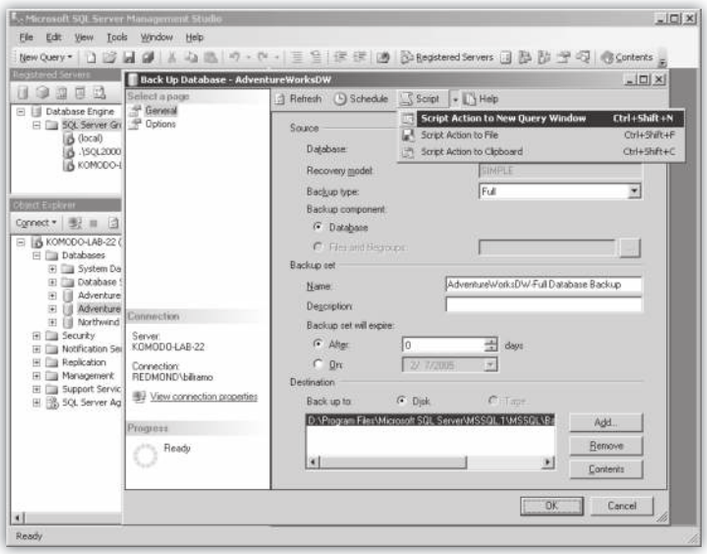


Figura 30.3. La interfaz de Management Studio de SQL Server.

30.2. Variaciones y extensiones de SQL

SQL Server permite a los desarrolladores de aplicaciones escribir la lógica corporativa del lado del servidor usando Transact-SQL o algún lenguaje de programación .NET, como C#, Visual Basic, COBOL o J++. Transact-SQL es un lenguaje de programación de bases de datos completo que incluye sentencias para la definición y la manipulación de los datos, sentencias iterativas y condicionales, variables, procedimientos y funciones. Transact-SQL soporta la mayor parte de las instrucciones y estructuras de consulta y de modificación de datos del LDD **obligatorias** de la norma SQL-2003. Consulte la Sección 30.2.1 para conocer la lista de tipos de datos de SQL-2003 soportados. Además de las características obligatorias, Transact-SQL también soporta muchas características **opcionales** de la norma SQL-2003, como las consultas recursivas, las expresiones comunes de las tablas, las funciones definidas por los usuarios y los operadores relacionales como **intersect** y **except,** entre otras.

30.2.1. Tipos de datos

SQL Server soporta todos los tipos de datos escalares obligatorios de la norma SQL-2003. SQL Server también soporta la posibilidad de dar otros nombres a los tipos del sistema usando nombres facilitados por los usuarios; el uso de alias es parecido en funcionalidad a los diferentes tipos de SQL-2003, pero no cumple completamente con ellos.

Entre los tipos primitivos exclusivos de SQL Server se encuentran:

- Los tipos de cadenas de caracteres y binarias de tamaño variable hasta los $2^{31} - 1$ bytes (**varchar/nvarchar/varbinary(max),** que tienen un modelo de programación similar a los tipos small character y byte string. Además, soportan un atributo de almacenamiento denominado FILESTREAM para especificar que los datos de cada valor de la columna se almacenan en un archivo separado en el sistema de archivos. El almacenamiento FILESTREAM permite un mayor rendimiento en el acceso a los datos para las aplicaciones que usan la API nativa del sistema de archivos.
- El tipo XML, descrito en la Sección 30.11, se usa para guardar datos de XML en las columnas de las tablas. El tipo XML puede tener, opcionalmente, una *colección de esquemas* de XML asociados que especifique la restricción de que los ejemplares de este tipo deben adherirse a uno de los tipos de XML definidos en la colección de esquemas.
- **sql_variant** es un tipo de datos escalares que puede contener valores de cualquier tipo escalar de SQL (excepto los tipos large character, large binary y **sql_variant**). Este tipo lo usan las aplicaciones que necesitan guardar datos cuyo tipo no se puede anticipar en el momento de definir los datos. **sql_variant** también es el tipo de las columnas formadas a partir de la ejecución del operador relacional **unpivot** (consulte la Sección 30.2.2). Internamente, el sistema realiza un seguimiento del tipo original de los datos. Es posible filtrar, reunir y ordenar las columnas sql_variant. La función del sistema **sql_variant_property** devuelve los detalles de los datos reales guardados en las columnas de tipo **sql_variant,** incluida la información sobre el tipo básico y sobre el tamaño.
- El tipo de dato **hierarchyId** facilita las tareas de almacenamiento y la realización de consultas de datos jerárquicos. Los datos jerárquicos se definen como conjuntos de datos relacionados entre sí mediante relaciones jerárquicas en las que un elemento de datos es el padre de otro. Ejemplos comunes son una estructura organizativa, un sistema de archivos jerárquico, un conjunto de tareas de un proyecto, las relaciones parte-subparte y un grafo de enlaces entre páginas web.

- SQL Server soporta el almacenamiento y la consulta de datos geoespaciales, es decir, datos referenciados a la Tierra. Los modelos comunes de estos datos son los sistemas de coordenadas planas y geodésicas. La principal diferencia entre ambas es que las últimas tienen en cuenta la curvatura de la Tierra. SQL Server soporta geometry y geography, que se corresponden con los modelos plano y geodésico.

Además, SQL Server soporta el tipo table y el tipo cursor, que no se pueden usar como columnas de las tablas, pero sí como variables del lenguaje Transact-SQL:

- El tipo **table** permite que una variable guarde un conjunto de filas. Los ejemplares de este tipo se usan, sobre todo, para guardar los resultados temporales de los procedimientos almacenados o como valor devuelto por funciones cuyo resultado es una tabla. Las variables table se comportan como variables locales. Tienen un ámbito bien definido, que es la función, el procedimiento almacenado o el procesamiento por lotes en el que se declaran. Dentro de su ámbito, las variables table se pueden utilizar como las tablas normales. Se pueden aplicar en cualquier lugar en el que se utilicen tablas o expresiones de tabla en las sentencias **select, insert, update** o **delete.**
- El tipo **cursor** permite las referencias a objetos cursor. El tipo cursor puede usarse para declarar variables, o argumentos de entrada/salida de rutina para referencia a cursores entre distintas llamadas de rutina.

30.2.2. Mejoras del lenguaje de consultas

Además de los operadores relacionales de SQL como la **reunión interna** y la **reunión externa,** SQL Server soporta los operadores relacionales **pivot, unpivot** y **apply.**

- **pivot** es un operador que transforma su conjunto de resultados de entrada con dos columnas, que representan pares nombre-valor, en varias columnas, una por cada nombre de la entrada. La columna de nombres de la entrada se denomina columna pivote. El usuario debe indicar los nombres que hay que transponer de la entrada a las diferentes columnas de la salida. Considere la tabla *VentasMensuales* (*IDProducto*, *Mes*, *CantidadVentas*). La consulta siguiente, que usa el operador **pivot,** devuelve la *CantidadVentas* de los meses de enero, febrero y marzo como columnas diferentes. Tenga en cuenta que el operador pivot también lleva a cabo una agregación implícita de todas las otras columnas de la tabla y una agregación explícita de la columna pivote.

```
select *
from VentasMensuales pivot(sum(CantidadVentas)
                           for Mes in ('Ene', 'Feb', 'Mar')) T;
```

La operación inversa a **pivot** es **unpivot.**

- El operador **apply** es similar a la función de reunión, excepto que la parte derecha es una expresión que contiene referencias a columnas de la parte izquierda, por ejemplo una invocación de función con valor que toma como argumento una o más columnas de su parte izquierda. Las columnas generadas por este operador son la unión de las columnas de sus dos entradas. El operador **apply** se puede usar para evaluar su entrada derecha para cada fila de su entrada izquierda y llevar a cabo una **union all** de las filas a lo largo de todas estas evaluaciones. Hay dos variedades del operador **apply** parecidas a las de **join,** es decir, **cross** y **outer.** Las dos variedades se diferencian en lo relativo al manejo en caso de que la entrada derecha produzca un conjunto de resultados vacío. En el caso de **cross apply,** hace que la fila correspondiente de la entrada izquierda no aparezca en el resultado. En el caso de **outer apply,** la fila de la entrada izquierda aparece con valores NULL para las columnas de la entrada derecha. Considere una función que se valora como tabla denominada *HallarInformes*, que toma como entrada el ID de cierto empleado y devuelve el conjunto de empleados de la organización que informa directa o indirectamente a ese empleado. La consulta siguiente invoca esta función para el jefe de cada departamento desde la tabla *Departamentos*:

```
select *
from Departamentos D cross apply
                     HallarInformes (D. IDJefe)
```

30.2.3. Rutinas

Los usuarios pueden escribir rutinas que se ejecuten dentro del proceso servidor como funciones escalares o tabulares, como procedimientos almacenados o como disparadores usando Transact-SQL o algún lenguaje .NET. Todas estas rutinas se definen para la base de datos mediante la instrucción **create** [**function, procedure, trigger**] del LDD. Las funciones escalares se pueden usar en cualquier expresión escalar de las instrucciones LMD o LDD de SQL. Las funciones que devuelven tablas se pueden utilizar en cualquier parte en que se permitan tablas en las instrucciones **select.** Las funciones que devuelven tablas de Transact-SQL cuyo cuerpo contiene una sola instrucción **select** de SQL se tratan como vistas (expandidas en línea) en la consulta que hace referencia a la función. Dado que las funciones que devuelven tablas permiten argumentos de entrada, las funciones en línea valoradas como tablas se pueden considerar vistas parametrizadas.

30.2.3.1. Vistas indexadas

Además de las vistas tradicionales definidas en la norma ANSI de SQL, SQL Server soporta las vistas indexadas (materializadas). Las vistas indexadas pueden mejorar sustancialmente el rendimiento de las consultas complejas de ayuda a la toma de decisiones que recuperan un importante número de filas y agregan grandes cantidades de información en sumas, recuentos y medias. SQL Server soporta la creación de índices agrupados en cada vista y, en consecuencia, cualquier número de índices no agrupados. Una vez indexada una vista, el optimizador puede usar sus índices en consultas que hagan referencia a la vista o a sus tablas base. No es necesario que las consultas se refieran explícitamente a las vistas indexadas para que se utilicen en el plan de consulta, ya que la asociación se realiza automáticamente por la definición de vista. De esta forma, las consultas ya existentes se pueden beneficiar de la eficiencia mejorada de recuperar los datos directamente de la vista indexada sin que haga falta reescribirlas. Las instrucciones de actualización de las tablas base de la vista se propagan automáticamente a las vistas indexadas.

30.2.4. Índices filtrados

Un índice filtrado es un índice optimizado no agrupado, especialmente apropiado para consultas de selección de un subconjunto de datos bien definido. Usa un predicado filtro para indexar una parte de las filas de la tabla. Un índice filtrado bien diseñado puede mejorar el rendimiento de las consultas, reducir el coste de mantenimiento del índice y reducir el coste de almacenamiento del índice comparado con los índices de tabla completa. Los índices filtrados pueden proporcionar las siguientes ventajas sobre los índices de tabla completa:

- **Mejora del rendimiento de las consultas y la calidad del plan.** Un índice filtrado bien diseñado mejora el rendimiento de las consultas y la calidad del plan de ejecución porque es de menor tamaño que un índice de tabla completa no agrupado y tiene estadísticas filtradas. Las estadísticas filtradas son más precisas que las de tabla completa, ya que solo cubren las filas del índice filtrado.

- **Reducción del coste de mantenimiento de índices.** Un índice solo se mantiene cuando los datos de las sentencias del lenguaje de manipulación de datos (LMD) afectan a los datos del índice. Un índice filtrado reduce el coste de mantenimiento del índice comparado con el de una tabla completa no agrupado, ya que es más pequeño y solo se mantiene cuando se ven afectados los datos del índice. Es posible tener un gran número de índices filtrados, especialmente cuando contienen datos que se ven raramente afectados. De forma similar, si un índice filtrado solo contiene datos que se ven raramente afectados, el menor tamaño del índice reduce el coste de actualización de las estadísticas.
- **Reducción del coste de almacenamiento de índices.** La creación de un índice filtrado puede reducir el almacenamiento de disco de índices no agrupados cuando el índice de tabla completa no es necesario. Se puede sustituir un índice de tabla completa no agrupado por varios índices filtrados sin un aumento significativo de los requisitos de almacenamiento.

También se pueden crear explícitamente estadísticas filtradas, independientemente de los índices filtrados.

30.2.4.1. Vistas actualizables y disparadores

Generalmente, las vistas pueden ser objetivo de las instrucciones **update, delete** o **insert** si la modificación de los datos solo se aplica a una de las tablas base de la vista. Las actualizaciones de las vistas divididas se pueden propagar a varias tablas base. Por ejemplo, la siguiente instrucción **update** incrementa los precios para el editor «0736» en un diez por ciento.

update *vistatítulos*
set *precio* = *precio* * 1.10
where *id_editorial* = '0736';

Para las modificaciones de datos que afectan a más de una tabla base, la vista se puede actualizar si hay algún disparador **instead** definido para la operación: los disparadores **instead** para las operaciones **insert, update** o **delete** se pueden definir sobre una vista para especificar las actualizaciones que hay que ejecutar en las tablas base con el objetivo de implementar las modificaciones de la vista correspondientes.

Los disparadores son procedimientos de Transact-SQL o de .NET que se ejecutan automáticamente cuando se envía una instrucción de LMD **(update, insert** o **delete)** a una tabla base o a una vista. Los disparadores son mecanismos que posibilitan la aplicación de la lógica corporativa de forma automática al modificar los datos o al ejecutar instrucciones de LDD. Los disparadores pueden extender la lógica de comprobación de la integridad de las restricciones declarativas, de las predeterminadas y de las reglas, aunque las restricciones declarativas se deben usar preferentemente siempre que sean suficientes, ya que se pueden usar por el optimizador de consultas para razonar sobre el contenido de los datos.

Los disparadores se pueden clasificar en LMD y en LDD, según el tipo de evento que los desencadene. Los disparadores LMD se definen contra tablas o vistas que se están modificando. Los disparadores LDD se definen contra bases de datos completas para una o varias instrucciones de LDD, como **create table, drop procedure,** etc.

Los disparadores se pueden clasificar en disparadores **after** e **instead,** según el momento en que se invocan en relación con la acción que los desencadena. Los disparadores **after** se ejecutan después de la instrucción de disparo y luego se aplican restricciones declarativas. Los disparadores **instead** se ejecutan en lugar de la acción que los dispara. Se puede considerar que los disparadores **instead** son parecidos a los disparadores **before,** pero sustituyen realmente a la acción de disparo. En SQL Server los disparadores **after** de LMD solo se pueden definir sobre las tablas base, mientras que los disparadores **instead** de LMD se pueden definir sobre las tablas base o sobre las vistas. Los disparadores **instead** permiten hacer actualizable prácticamente cualquier vista mediante lógica que proporciona el usuario. Los disparadores **instead** LDD se pueden definir sobre cualquier sentencia del LDD.

30.3. Almacenamiento e índices

En SQL Server, una base de datos hace referencia a un conjunto de archivos que contienen datos y están soportados por un único registro histórico de transacciones. La base de datos es la unidad principal de administración de SQL Server, y también proporciona un contenedor para estructuras físicas como las tablas y los índices y para estructuras lógicas como las restricciones, las vistas, etc.

30.3.1. Grupos de archivos

Con el fin de gestionar el espacio de la base de datos de forma efectiva, el conjunto de archivos de datos de la base de datos se divide en grupos denominados grupos de archivos. Cada grupo de archivos contiene uno o más archivos del sistema operativo.

Cada base de datos tiene, al menos, un grupo de archivos conocido como grupo de archivos principal. Este grupo de archivos contiene todos los metadatos de la base de datos en tablas del sistema. El grupo de archivos principal también puede contener datos de usuario.

Si se crean grupos de archivos adicionales definidos por el usuario, este puede controlar de forma explícita la ubicación de cada tabla, de cada índice o de cada columna de objetos de gran tamaño de las tablas colocándolas en un grupo de archivos. Por ejemplo, el usuario puede decidir guardar los índices críticos para el rendimiento en un grupo de archivos en discos de estado sólido. En otro caso, puede decidir situar columnas varbinary(max) que contengan datos de vídeo en subsistemas de E/S optimizados para el streaming.

30.3.2. Gestión del espacio en los grupos de archivos

Uno de los principales propósitos de los grupos de archivos es permitir una gestión efectiva del espacio. Todos los archivos de datos se dividen en unidades de tamaño fijo de ocho kilobytes denominadas **páginas.** El sistema de asignación es responsable de asignar esas páginas a las tablas y a los índices. El objetivo del sistema de asignación es minimizar la cantidad de espacio desperdiciado y, al tiempo, mantener el grado de fragmentación de la base de datos al mínimo para garantizar un buen rendimiento de exploración. Con el fin de lograr este objetivo, el administrador de asignación suele asignar y extraer todas las páginas en grupos de ocho páginas contiguas denominadas **extensiones.**

El sistema de asignación gestiona estas extensiones mediante varios mapas de bits. Estos mapas de bits permiten al sistema de asignación encontrar una página o una extensión para asignarla de forma rápida. Estos mapas de bits también se usan cuando se ejecuta una exploración de tabla completa o de índices. La ventaja de usar mapas de bits basados en la asignación para la exploración es que permite recorrer en el orden del disco todas las extensiones que pertenecen al nivel de hojas de la tabla o del índice, lo que mejora significativamente el rendimiento de la exploración.

Si hay más de un archivo en un grupo de archivos, el sistema de asignación asigna extensiones para cualquier objeto de ese grupo de archivos mediante un algoritmo de «relleno proporcional». Cada archivo se rellena en proporción a la cantidad de espacio libre de ese archivo en comparación con los demás archivos. Esto rellena todos los archivos del grupo de archivos aproximadamente al mismo ritmo, y permite al sistema usar por igual todos los archivos

del grupo de archivos. Los archivos se pueden configurar para que crezcan automáticamente si el grupo de archivos se está quedando sin espacio. SQL Server permite que los archivos disminuyan de tamaño. Para disminuir el tamaño de un archivo de datos, SQL Server traslada todos los datos desde el extremo físico del archivo a un punto más cercano al inicio del archivo y luego reduce realmente su tamaño, devolviendo el espacio liberado al sistema operativo.

30.3.3. Tablas

SQL Server soporta las organizaciones de las tablas en montones (o montículos: *heap*) y en agrupaciones (*clusters*). En las tablas organizadas en montículos la ubicación de cada fila de la tabla la determina completamente el sistema y el usuario no la especifica en absoluto. Las filas de los montículos tienen un identificador fijo conocido como identificador de la fila (*row identifier*: RID), y su valor no cambia nunca, a no ser que se reduzca el tamaño del archivo y la fila se traslade. Si la fila se hace tan grande que no cabe en la página en la que se insertó originalmente, el registro se traslada a un lugar distinto, pero se deja un resguardo de entrega en la ubicación original, de modo que el registro todavía se pueda encontrar usando su RID original.

En la organización de índices agrupados de las tablas, las filas de la tabla se guardan en un árbol B^+ ordenado por la clave de agrupamiento del índice. La clave del índice agrupado también sirve como identificador único de cada fila. La clave del índice agrupado se puede definir como no única, en cuyo caso SQL Server agrega una columna oculta adicional para hacer que la clave sea única. El índice agrupado también sirve como estructura de búsqueda para identificar una fila de la tabla con una clave concreta o explorar un conjunto de filas de la tabla con las claves ubicadas dentro de un cierto rango. Los índices agrupados son el tipo más frecuente de organización de las tablas.

30.3.4. Índices

SQL Server también soporta los índices de árbol B^+ secundarios (no agrupados). Las consultas que solo hacen referencia a las columnas que están disponibles a través de índices secundarios se procesan mediante la recuperación de las páginas desde el nivel hoja de los índices sin necesidad de recuperar los datos del índice agrupado o del montículo. Los índices no agrupados de las tablas con índices agrupados contienen las columnas clave del índice agrupado. Por tanto, las filas del índice agrupado se pueden trasladar a una página diferente (mediante divisiones, desfragmentaciones o recreaciones del índice) sin necesidad de realizar modificaciones en los índices no agrupados.

SQL Server soporta la adición de columnas calculadas a las tablas. Las columnas calculadas son columnas cuyo valor es una expresión, normalmente basada en el valor de otras columnas de esa fila. SQL Server permite que el usuario construya índices secundarios en términos de las columnas calculadas.

30.3.5. Particiones

SQL Server soporta la división de las tablas y de los índices no agrupados. Los índices divididos están constituidos por varios árboles B^+, uno por partición. Las tablas divididas sin índices (montículos) están constituidas por varios montículos, uno por partición. Por brevedad, a partir de aquí solo se hará referencia a los índices divididos (agrupados o sin agrupar) y se ignorarán los montículos.

La división de los índices de gran tamaño permite al administrador mayor flexibilidad en la gestión del almacenamiento del índice y puede mejorar el rendimiento de algunas consultas, ya que las particiones actúan como índices de grano grueso.

La división de los índices se especifica proporcionando una función y un esquema de partición. La función de partición asigna el dominio de la columna de partición (cualquier columna del índice) a las particiones numeradas de 1 a *N*. El esquema de la partición asigna los números de las particiones generadas por la función de partición a grupos de archivos concretos en los que se guardan las particiones.

30.3.6. Creación en línea de índices

La creación de índices nuevos y la reconstrucción de los ya existentes en una tabla se puede llevar a cabo en línea, es decir, mientras se están realizando las operaciones de selección, inserción, borrado o actualización en esa tabla. La creación del nuevo índice tiene lugar en tres fases. La primera fase no es más que la creación de un árbol B^+ vacío para el nuevo índice con el catálogo que muestre que el nuevo índice está disponible para operaciones de mantenimiento. Es decir, todas las operaciones posteriores de inserción, borrado o actualización deben mantener el nuevo índice, pero este no se halla disponible para las consultas. La segunda fase consiste en la exploración de la tabla para recuperar las columnas del índice de cada fila, ordenar las filas e insertarlas en el nuevo árbol B^+. Estas inserciones deben tener cuidado al interactuar con las otras filas del nuevo árbol B^+ colocadas allí por las operaciones de mantenimiento del índice debidas a las actualizaciones de la tabla base. La exploración es de instantáneas, sin bloqueos, y garantiza que la exploración vea toda la tabla únicamente con los resultados de las transacciones comprometidas en el momento de comenzar la exploración. Esto se consigue usando la tecnología de aislamiento de instantáneas descrita en la Sección 30.5.1. La fase final de la creación del índice supone la actualización del catálogo para que indique que la creación del índice se ha completado y que este se encuentra disponible para las consultas.

30.3.7. Exploraciones y lecturas anticipadas

La ejecución de las consultas en SQL Server puede involucrar varios modos de exploración diferentes de las tablas y de los índices subyacentes. Entre estos modos de exploración están las exploraciones ordenadas y las desordenadas, las exploraciones en serie y las paralelas, las unidireccionales y las bidireccionales, las exploraciones hacia delante y hacia atrás, la exploración de toda la tabla o de todo el índice y las exploraciones de rango o filtradas.

Cada uno de estos modos de exploración tiene un mecanismo de lectura anticipada que intenta que la exploración se anticipe a las necesidades de ejecución de la consulta, con el fin de reducir las sobrecargas de búsqueda y de latencia y usar el tiempo de disco no ocupado. El algoritmo de lectura anticipada de SQL Server usa el conocimiento del plan de ejecución de la consulta con el fin de conducir la lectura anticipada y asegurarse de que solamente se lean los datos que la consulta necesita realmente. Además, la cantidad de lectura anticipada se dimensiona de forma automática según el tamaño de la memoria intermedia agrupada, el volumen de E/S que el subsistema del disco puede sostener y la velocidad a la que la ejecución de la consulta consume los datos.

30.3.8. Compresión

SQL Server soporta la compresión tanto de *filas* como de *páginas* para tablas e índices. La compresión de filas utiliza un formato de longitud variable para tipos de datos como enteros y que tradicionalmente son considerados de tamaño fijo. La compresión de páginas elimina los prefijos comunes de las columnas y construye un diccionario por página para los valores comunes.

30.4. Procesamiento y optimización de consultas

El procesador de consultas de SQL Server está basado en un entorno extensible que permite la rápida incorporación de nuevas técnicas de ejecución y de optimización. Cualquier consulta de SQL se puede expresar en forma de árbol de operadores del álgebra relacional extendida. Mediante la abstracción de los operadores de este álgebra en **iteradores,** la ejecución de la consulta encapsula los algoritmos de procesamiento de datos como unidades lógicas que se comunican entre sí usando la interfaz GetNextRow(). A partir del árbol inicial de la consulta, el optimizador de consultas de SQL Server genera alternativas utilizando transformaciones en árbol y estima su coste de ejecución teniendo en cuenta el comportamiento de los iteradores y los modelos estadísticos para estimar el número de filas a procesar.

30.4.1. Visión general del proceso de optimización

Las consultas complejas presentan oportunidades significativas de optimización que exigen la ordenación de los operadores de unos bloques de consulta a otros y la selección de los planes con base únicamente en los costes estimados. Para aprovechar estas oportunidades, el optimizador de consultas de SQL Server se desvía de los enfoques tradicionales de la optimización de consultas usados en otros sistemas comerciales en favor de un entorno más general, puramente algebraico, que se basa en el prototipo de optimizador Cascades. La optimización de las consultas forma parte de su proceso de compilación, que consta de cuatro pasos:

- **Análisis/vinculación.** Tras el análisis, el vinculador resuelve los nombres de tablas y columnas mediante los catálogos. SQL Server usa una caché de plan para evitar repetir la optimización de consultas idénticas o estructuralmente parecidas. Si no se dispone de plan guardado en la caché, se genera un árbol de operadores inicial. El árbol de operadores no es más que una combinación de operadores relacionales y no está restringido por conceptos como los bloques de consulta o las tablas derivadas, que suelen obstaculizar la optimización.
- **Simplificación/normalización.** El optimizador aplica las reglas de simplificación al árbol de operadores para obtener una forma normal y simplificada. Durante la simplificación, el optimizador determina y carga las estadísticas necesarias para la estimación de la cardinalidad.
- **Optimización basada en el coste.** El optimizador aplica las reglas de exploración y de implementación para generar alternativas, estimar el coste de ejecución y escoger el plan con el coste anticipado más bajo. Las reglas de exploración implementan la reordenación de un amplio conjunto de operadores, incluida la reordenación de la reunión y de la agregación. Las reglas de implementación introducen alternativas de ejecución como las reuniones por mezcla y las reuniones por asociación.
- **Preparación del plan.** El optimizador crea las estructuras del plan de ejecución para el plan seleccionado.

Para obtener mejores resultados, la optimización basada en el coste de SQL Server no se divide en fases que optimicen distintos aspectos de la consulta por separado; además, no está restringida a una sola dimensión, como puede ser la enumeración de reuniones. En su lugar, un conjunto de reglas de transformación define el espacio de interés, y la estimación del coste se usa de manera uniforme para seleccionar un plan eficiente.

30.4.2. Simplificación de las consultas

Durante la simplificación solo se aplican las transformaciones que garantizan la generación de sustitutos menos costosos. El optimizador envía las selecciones tan abajo del árbol de operadores como sea posible; comprueba los predicados en búsqueda de contradicciones, teniendo en cuenta las restricciones declaradas. Usa las contradicciones para identificar subexpresiones que se puedan eliminar del árbol. Una situación frecuente es la eliminación de las ramas **union** que recuperan los datos de las tablas con diferentes restricciones.

Una serie de reglas de simplificación son *dependientes del contexto*; es decir, la sustitución solamente es válida en el contexto de uso de la subexpresión. Por ejemplo, una reunión externa se puede simplificar en reunión interna si una operación de filtrado posterior elimina las reglas no coincidentes que se rellenaron con **null.** Otro ejemplo es la eliminación de las reuniones sobre las claves externas, que no hace falta ejecutar si no existe un uso posterior de las columnas de la tabla a la que se hace referencia. Un tercer ejemplo es el contexto de insensibilidad a los duplicados, que indica que la entrega de una o más copias de una fila no afecta al resultado de la consulta. Las subexpresiones bajo las semirreuniones y bajo **distinct** son insensibles a los duplicados, lo que permite cambiar **union** por **union all,** por ejemplo.

Para la agrupación y la agregación se usa el operador *GbAgg*, que crea grupos y, opcionalmente, aplica una función agregada a cada grupo. La eliminación de duplicados, expresada en SQL mediante la palabra clave **distinct,** es sencillamente un *GbAgg* sin funciones agregadas que calcular. Durante la simplificación, la información sobre las claves y sobre las dependencias funcionales se usa para reducir el agrupamiento de columnas.

Las subconsultas se normalizan mediante la eliminación de las especificaciones de consulta correlacionadas y el uso de algunas variantes de la reunión en su lugar. La eliminación de las correlaciones no es una «estrategia de ejecución de subconsultas», sino simplemente un paso de la normalización. Luego se consideran una serie de estrategias de ejecución, durante la optimización basada en el coste.

30.4.3. Reordenación y optimización basadas en el coste

En SQL Server las transformaciones se integran completamente en la generación basada en el coste y en la selección de los planes de ejecución. El optimizador de consultas de SQL Server incluye alrededor de 350 reglas de transformación lógica y física. Además de la reordenación de la reunión interna, el optimizador de consultas usa transformaciones de reordenación para los operadores reunión externa, semirreunión y antisemirreunión del álgebra relacional estándar (con duplicados para SQL). También se reordena *GbAgg*, trasladándolo por debajo de las reuniones siempre que sea posible. La agregación parcial; es decir, la introducción de un nuevo *GbAgg* con agrupación sobre un superconjunto de las columnas de un *GbAgg* posterior, se considera por debajo de las reuniones y de **union all,** y también en los planes paralelos. Consulte las referencias dadas en las notas bibliográficas para obtener más detalles.

Durante la exploración del plan se considera la ejecución correlacionada; el caso más simple es una reunión de búsqueda en el índice. SQL Server modela la ejecución correlacionada como un operador algebraico único, denominado **apply,** que opera sobre la tabla T y la expresión relacional parametrizada $E(t)$. **Apply** ejecuta E para cada fila de T, que proporciona los valores de los parámetros. La ejecución correlacionada se considera como una alternativa a la ejecución, independientemente del uso de subconsultas en la formulación original de SQL. Es una estrategia muy eficiente cuando la tabla T es muy pequeña y los índices soportan la ejecución parametrizada eficiente de $E(t)$. Además, se considera la reducción del número de ejecuciones de $E(t)$ cuando hay valores duplicados de los parámetros mediante dos técnicas: ordenar T según el valor

de los parámetros, de forma que se reutilice un único resultado de $E(t)$ mientras que el valor del parámetro sigue siendo el mismo, o usar una tabla de asociación que realice un seguimiento del resultado de $E(t)$ para (algún subconjunto de) los valores anteriores del parámetro.

Algunas aplicaciones seleccionan las filas según algún resultado agregado obtenido para su grupo. Por ejemplo, «Encontrar los clientes cuyo saldo sea mayor que el doble de la media de su segmento de mercado». La formulación de SQL exige una autorreunión. Durante la exploración se detecta este patrón y se considera la ejecución por segmentos como alternativa a la autorreunión.

También se considera el uso de vistas materializadas durante la optimización basada en el coste. Las interacciones de la coincidencia de vistas con la ordenación de operadores en ese uso puede que no resulte evidente hasta que haya tenido lugar otra reordenación. Cuando se encuentra que una vista coincide con alguna subexpresión, la tabla que contiene el resultado de esa vista se agrega como alternativa a la expresión correspondiente. En función de la distribución de los datos y de los índices disponibles, puede que sea mejor que la expresión original; la selección se realizará en términos de la estimación del coste.

Para estimar el coste de ejecución del plan, el modelo tiene en cuenta el número de veces que se ejecuta una subexpresión, así como el **objetivo de filas,** que es el número de filas que se espera consumir por el operador padre. El objetivo de filas puede ser menor que la estimación de la cardinalidad en casos de consultas de n-mayores y por *Apply/semijoin*. Por ejemplo, *Apply/semijoin* devuelve la fila t de T tan pronto como $E(t)$ produce una fila (es decir, comprueba que $E(t)$ **existe**). Por tanto, el objetivo de filas del resultado de $E(t)$ es 1, y los objetivos de filas de los subárboles de $E(t)$ se calculan para este objetivo de filas de $E(t)$ y se usan para la estimación del coste.

30.4.4. Planes de actualización

Los planes de actualización optimizan el mantenimiento de índices, comprueban las restricciones, aplican las acciones en cascada y mantienen las vistas materializadas. Para el mantenimiento de los índices, en lugar de tomar cada fila y mantener todos sus índices, los planes de actualización aplican las modificaciones índice a índice, ordenando las filas y aplicando la operación **update** según el orden de la clave. Esto minimiza las operaciones aleatorias de E/S, especialmente cuando el número de filas que hay que actualizar es grande. Las restricciones las maneja un operador **assert,** que ejecuta un predicado y envía un mensaje de error si el resultado es **false.** Las restricciones de integridad referencial se definen mediante predicados **exist** que, a su vez, se convierten en semirreuniones y se optimizan considerando todos los algoritmos de ejecución.

El problema de Halloween (descrito anteriormente en la Sección 13.6) hace referencia a la siguiente anomalía: suponga que se lee un índice salarial en orden ascendente y se están subiendo los sueldos un diez por ciento. Como resultado de la actualización, las filas se desplazarán hacia arriba en el índice, se volverán a encontrar y se actualizarán de nuevo, lo que lleva a un bucle infinito. Una forma de abordar este problema es separar el procesamiento en dos fases: en primer lugar se leen todas las filas que se van a actualizar y se hace una copia de ellas en algún emplazamiento temporal, después se leen desde ese emplazamiento y se aplican todas las actualizaciones. Otra alternativa es leer desde un índice distinto en el que las filas no se trasladen como consecuencia de la actualización. Algunos planes de ejecución proporcionan la separación de las fases de forma automática, si se ordena o crea una tabla de asociación con las filas que se van a actualizar. En el optimizador de SQL Server la protección contra Halloween se modela como una de las propiedades de los planes. Se generan varios planes que proporcionan la propiedad requerida y se selecciona uno en función del coste de ejecución estimado.

30.4.5. Análisis de los datos durante la optimización

SQL fue de los primeros en introducir técnicas para llevar a cabo la recogida de estadísticas como parte de las optimizaciones en proceso. El cálculo de las estimaciones de tamaño del resultado se basa en las estadísticas de las columnas usadas en una expresión dada. Estas estadísticas consisten en histogramas de diferencias máximas de los valores de las columnas y en varios contadores que capturan la densidad y el tamaño de las filas, entre otras cosas. Los administradores de las bases de datos pueden crear estadísticas de manera explícita usando la sintaxis extendida de SQL.

Si no se dispone de estadísticas para una columna determinada, el optimizador de SQL Server detiene no obstante la optimización en proceso y reúne las estadísticas necesarias. En cuanto se han calculado las estadísticas, se reanuda la optimización original, que aprovechará las estadísticas recién creadas. La optimización de las consultas posteriores reutiliza las estadísticas generadas anteriormente. Normalmente, tras un breve periodo de tiempo, ya se han creado las estadísticas de las columnas usadas con frecuencia y las interrupciones para la elaboración de estadísticas nuevas se hacen menos frecuentes. Mediante el seguimiento del número de filas modificadas en cada tabla se tiene una medida de la antigüedad de todas las estadísticas afectadas. Una vez que la antigüedad supera un cierto umbral, se vuelven a calcular esas estadísticas y los planes guardados en la caché se vuelven a compilar para que tengan en cuenta las distribuciones de datos modificadas.

Las estadísticas se pueden recalcular de manera asíncrona, lo que evita tiempos de compilación potencialmente largos generados por cálculos síncronos. La optimización que desencadena el cálculo de las estadísticas usa estadísticas potencialmente antiguas. No obstante, las consultas posteriores pueden aprovechar las estadísticas recalculadas. Esto permite lograr un equilibrio aceptable entre el tiempo empleado en la optimización y la calidad del plan de consultas resultante.

30.4.6. Búsquedas parciales y heurísticas

Los optimizadores basados en el coste se enfrentan con el problema de la explosión del espacio de búsqueda, puesto que las aplicaciones realizan consultas que implican a docenas de tablas. Para abordar esto, SQL Server utiliza varias etapas de optimización, cada una de las cuales usa transformaciones de la consulta para explorar regiones sucesivamente mayores del espacio de búsqueda.

Hay transformaciones sencillas y completas diseñadas para la optimización exhaustiva, así como transformaciones inteligentes que implementan varias heurísticas. Las transformaciones inteligentes generan planes que están muy lejos entre sí en el espacio de búsqueda, mientras que las transformaciones sencillas exploran zonas vecinas. Las etapas de optimización aplican una mezcla de ambas clases de transformaciones, poniendo en primer lugar el énfasis en las transformaciones inteligentes y pasando luego a las transformaciones sencillas. Se conservan los resultados óptimos de los subárboles, de forma que las etapas posteriores puedan aprovechar los resultados generados con anterioridad. Cada etapa debe equilibrar las distintas técnicas de generación de planes:

- **Generación exhaustiva de alternativas.** Para generar el espacio completo el optimizador usa transformaciones completas, locales, no redundantes; una regla de transformación equivalente a una secuencia de transformaciones más primitivas solamente introduce una sobrecarga adicional.

- **Generación heurística de candidatos.** Es probable que una serie de candidatos interesantes (seleccionados en términos del coste estimado) estén lejos entre sí en términos de las reglas de transformación primitivas. En este caso, las transformaciones deseadas son incompletas, globales y redundantes.

La optimización se puede terminar en cualquier momento tras la generación del primer plan. Esta terminación se basa en el coste estimado del mejor plan encontrado y en el tiempo ya empleado en la optimización. Por ejemplo, si una consulta solo necesita buscar unas cuantas filas de algunos índices, se producirá rápidamente un plan muy barato en las primeras etapas, lo que terminará la optimización. Este enfoque permite agregar fácilmente nuevas heurísticas con el transcurso del tiempo, sin comprometer la selección de planes basada en el coste ni la exploración exhaustiva del espacio de búsqueda, cuando resulte conveniente.

30.4.7. Ejecución de las consultas

Los algoritmos de ejecución soportan tanto el procesamiento basado en la ordenación como el basado en la asociación, y sus estructuras de datos se diseñan para optimizar el uso de la caché del procesador. Las operaciones de asociación soportan la agregación y la reunión básicas, con una serie de optimizaciones, extensiones y ajustes dinámicos del sesgo de datos. La operación **flow-distinct** es una variante diferente de la asociación (*hash distinct*), en la que las filas se devuelven antes, tan pronto como se encuentra un nuevo valor diferente, en lugar de esperar a procesar todos los datos de entrada. Este operador es efectivo para las consultas que usan **distinct** y solo piden unas cuantas filas, por ejemplo, cuando se usa el constructor **top n.** Los planes correlacionados especifican la ejecución de $E(t)$, e incluyen a menudo varias búsquedas en el índice basadas en el parámetro, para cada fila t de la tabla T. La **captura previa asíncrona** permite la solicitud al motor de almacenamiento de varias solicitudes de búsqueda en el índice. Se implementa de esta manera: se realiza una solicitud de búsqueda en el índice sin bloqueo para la fila t de T, luego t se coloca en la cola de captura previa. Las filas se sacan de la cola y **apply** las usa para ejecutar $E(t)$. La ejecución de $E(t)$ no necesita que los datos se encuentren ya en la memoria intermedia agrupada, pero contar con buenas operaciones de captura previa maximiza el uso del hardware e incrementa el rendimiento. El tamaño de la cola se determina dinámicamente como función de los aciertos de la caché. Si no se necesita ninguna ordenación de las filas de salida de **apply,** las filas de esa cola se pueden tomar sin prestar atención al orden, para minimizar la espera en las operaciones de E/S.

La ejecución en paralelo la implementa el operador **exchange,** que gestiona varias hebras, realiza particiones o difunde datos y proporciona estos a varios procesos. El optimizador de consultas decide la ubicación de **exchange** según el coste estimado. El grado de paralelismo se determina dinámicamente en el momento de la ejecución, en función del uso del sistema en ese momento.

Los planes de índices están constituidos por los fragmentos que se han descrito anteriormente. Por ejemplo, se considera el uso de una reunión de índices para resolver las conjunciones de predicados (o una unión de índices para las disyunciones), en términos de su coste. Esta reunión se puede realizar en paralelo, usando cualquiera de los algoritmos de reunión de SQL Server. También se consideran reuniones de índices con el único propósito de ensamblar una fila con el conjunto de columnas necesario en una consulta, lo cual a veces es más rápido que explorar una tabla base. Tomar los identificadores de registros de un índice secundario y localizar la fila correspondiente de la tabla base es equivalente, en efecto, a ejecutar una reunión de búsqueda de índices. Para ello se usan las técnicas genéricas de ejecución correlacionada, como la captura previa asíncrona.

La comunicación con el motor de almacenamiento se realiza mediante OLE-DB, que permite el acceso a otros proveedores de datos que también implementan esa interfaz. OLE-DB es el mecanismo utilizado para las consultas distribuidas y remotas, que maneja directamente el procesador de consultas. Los proveedores de datos se clasifican según el rango de funcionalidad que proporcionan, desde simples proveedores de conjuntos de filas sin capacidades de indexado a proveedores con soporte completo de SQL.

30.5. Concurrencia y recuperación

Los subsistemas de transacciones, registro histórico, bloqueos y recuperación de SQL Server hacen que se cumplan las propiedades ACID esperadas de los sistemas de bases de datos.

30.5.1. Transacciones

En SQL Server todas las sentencias son atómicas y las aplicaciones pueden especificar distintos niveles de aislamiento para cada sentencia. Una transacción puede incluir sentencias que no solo seleccionen, inserten, borren o actualicen registros, sino también creen o eliminen tablas, construyan índices e importen datos en bruto. Las transacciones pueden extenderse por varias bases de datos en servidores remotos. Cuando las transacciones son sobre varios servidores, SQL Server utiliza un servicio del sistema operativo Windows llamado Coordinador de transacciones distribuidas de Microsoft (*Microsoft Distributed Transaction Coordinator*: MS DTC) para realizar el procesamiento de compromiso de dos fases. MS DTC soporta el protocolo de transacción XA y, junto con OLE-DB, proporciona la base para las transacciones ACID entre sistemas heterogéneos.

El control de concurrencia basado en los bloqueos es el predeterminado para SQL Server. SQL Server también ofrece el control de concurrencia optimista para los cursores. El control de concurrencia optimista se basa en la suposición de que los conflictos de recursos entre varios usuarios son poco probables (aunque no imposibles) y permite que las transacciones se ejecuten sin bloquear ningún recurso. Solo cuando se intenta modificar los datos, SQL Server comprueba los recursos para determinar si se ha producido algún conflicto. Si se produce algún conflicto, la aplicación debe leer los datos e intentar el cambio de nuevo. Las aplicaciones pueden elegir si se detectan los cambios comparando los valores o comprobando la columna especial de versión de fila (*row version*) de cada fila.

SQL Server soporta los niveles de aislamiento de SQL de lectura no comprometida, lectura comprometida, lectura repetible y serializable. La lectura comprometida es el nivel predeterminado. Además, SQL Server soporta dos niveles de aislamiento basados en las instantáneas (el aislamiento de instantáneas se describió anteriormente en la Sección 15.7).

- **Instantánea.** Especifica que los datos leídos por cualquier instrucción de la transacción son la versión consistente transaccionalmente de los datos que existían al comienzo de esa transacción. El efecto es como si las sentencias de una transacción viesen una instantánea de los datos que se comprometieron como si hubieran existido antes del comienzo de la transacción. Las escrituras se validan utilizando los pasos de validación descritos en la Sección 15.7 y se permite su terminación si la validación se cumple.
- **Instantánea de lectura comprometida.** Especifica que cada instrucción ejecutada en la transacción ve una instantánea transaccionalmente consistente de los datos tal y como eran al comienzo de la sentencia. Esto es diferente de lo que ocurre con el aislamiento de lectura comprometida, en el que la instrucción puede ver las actualizaciones comprometidas de las transacciones que se comprometen mientras se ejecuta.

30.5.2. Bloqueos

El bloqueo es el principal mecanismo utilizado para hacer cumplir la semántica de los niveles de aislamiento. Todas las actualizaciones adquieren los bloqueos exclusivos suficientes, mantenidos durante toda la transacción, para evitar que se produzcan actualizaciones que entren en conflicto entre sí. Los bloqueos compartidos se mantienen en duraciones diferentes para proporcionar los diversos niveles de aislamiento de SQL para las consultas.

SQL Server proporciona bloqueos de varias granularidades que permiten que cada transacción bloquee diferentes tipos de recursos (consulte la Figura 30.4, en la que los recursos se relacionan en orden creciente de granularidad). Para minimizar el coste de los bloqueos, SQL Server bloquea de manera automática los recursos según la granularidad apropiada para cada tarea. El bloqueo con una granularidad menor, como pueden ser las filas, aumenta la concurrencia, pero tiene una sobrecarga mayor, ya que hay que realizar más bloqueos si se bloquean muchas filas.

Los modos de bloqueo fundamentales de SQL Server son el compartido (S: *shared*), el de actualización (U: *update*) y el exclusivo (X: *exclusive*); los bloqueos intencionales (*intent*) se usan también para el bloqueo de varias granularidades. Los bloqueos de actualización se utilizan para evitar una forma frecuente de interbloqueo que se produce cuando varias sesiones leen, bloquean y, potencialmente, actualizan posteriormente los recursos. Los modos adicionales de bloqueo —denominados bloqueos de rango de claves— solo se toman en el nivel de aislamiento serializable para bloquear el rango entre dos filas de un índice.

Recurso	*Descripción*
RID	Identificador de fila, usado para bloquear una sola fila de la tabla
Clave	Bloqueo de fila en un índice; protege rangos de la clave en transacciones secuenciables
Página	Página de tabla o de índice de ocho kilobytes
Extensión	Grupo contiguo de ocho páginas de datos o de índices
Tabla	Tabla completa, incluidos todos los datos y todos los índices
BD	Base de datos

Figura 30.4. Recursos bloqueables.

30.5.2.1. Bloqueos dinámicos

Los bloqueos de granularidad fina pueden mejorar la concurrencia a cambio de ciclos adicionales de CPU y de memoria extra para adquirir y mantener muchos bloqueos. Cuando existen muchas consultas, una granularidad de bloqueo más gruesa proporciona un mejor rendimiento sin pérdida (o con una pérdida mínima) de la concurrencia. Los sistemas de bases de datos tradicionalmente han exigido sugerencias de consulta y opciones de tabla para que las aplicaciones especificaran la granularidad del bloqueo. Además, hay parámetros de configuración (frecuentemente estáticos) para la cantidad de memoria que se debe dedicar al administrador de bloqueos.

En SQL Server la granularidad del bloqueo se optimiza automáticamente para un rendimiento y una concurrencia óptimos de cada índice de la consulta. Además, la memoria dedicada al administrador de bloqueos se ajusta dinámicamente según la realimentación de las demás partes del sistema, incluidas otras aplicaciones de la máquina.

La granularidad del bloqueo se optimiza antes de la ejecución de las consultas para cada tabla e índice usados en esa consulta. El proceso de optimización del bloqueo tiene en cuenta el nivel de aislamiento (esto es, el tiempo que se mantienen los bloqueos), el tipo de exploración (rango, sonda o toda la tabla), el número estimado de filas que hay que explorar, la selectividad (porcentaje de filas visitadas que son aceptables para la consulta), la densidad de las filas (número de filas por página), el tipo de operación (exploración, actualización), los límites del usuario sobre la granularidad y la memoria de sistema disponible.

Una vez se está ejecutando la consulta, la granularidad de bloqueo se **dimensiona** automáticamente hasta el nivel de las tablas si el sistema adquiere significativamente más bloqueos de los esperados por el optimizador, o si la cantidad de memoria disponible baja y no se puede soportar el número de bloqueos necesario.

30.5.2.2. Detección de interbloqueos

SQL Server detecta de forma automática los interbloqueos que involucran tanto a bloqueos como a otros recursos. Por ejemplo, si la transacción A mantiene un bloqueo sobre Tabla1 y está esperando que haya memoria disponible y la transacción *B* tiene algo de memoria que no puede liberar hasta que adquiera un bloqueo sobre Tabla1, las transacciones sufrirán un interbloqueo. Las hebras y las memorias intermedias de comunicación también pueden estar implicadas en los interbloqueos. Cuando SQL Server detecta un interbloqueo, elige como víctima del interbloqueo la transacción que es menos costosa de hacer retroceder, considerando la cantidad de trabajo que la transacción ya ha realizado.

La detección frecuente de interbloqueos puede perjudicar al rendimiento del sistema. SQL Server ajusta automáticamente la frecuencia de detección de los interbloqueos a la frecuencia con la que se producen. Si los interbloqueos no son frecuentes, el algoritmo de detección se ejecuta cada cinco segundos. Si son frecuentes, comenzará a comprobar si hay alguno cada vez que una transacción espere un bloqueo.

30.5.2.3. Versiones de las filas para el aislamiento de instantáneas

Los dos niveles de aislamiento basados en las instantáneas usan las versiones de las filas para conseguir el aislamiento de las consultas sin bloquear las consultas que se hallan tras las actualizaciones, y viceversa. Bajo el aislamiento de instantáneas, las operaciones de actualización y de borrado generan versiones de las filas afectadas y las guardan en una base de datos temporal. El sistema se deshace de estas versiones cuando no hay ninguna transacción activa que pueda necesitarlas. Por tanto, las consultas ejecutadas bajo el aislamiento de instantáneas no necesitan adquirir bloqueos y, en vez de eso, pueden leer las versiones más antiguas de cualquier registro que otra transacción actualice o borre. Las versiones de las filas se usan también para proporcionar instantáneas de tablas para las operaciones de creación de índices en línea.

30.5.3. Recuperación y disponibilidad

SQL Server está diseñado para recuperarse de los fallos del sistema y de los medios, y el sistema de recuperación se puede adaptar a máquinas con grupos de memorias intermedias de tamaño muy grande (cien gigabytes) y miles de unidades de disco.

30.5.3.1. Recuperación de caídas

El registro histórico es, desde un punto de vista lógico, una corriente potencialmente infinita de registros históricos identificados por los números de secuencia del registro histórico (*log sequence numbers*: LSN). Desde un punto de vista físico, parte del flujo de registros se almacena en los archivos del registro histórico. Los registros históricos se guardan en los archivos del registro histórico hasta que se realiza una copia de seguridad y el sistema ya no los necesita para el retroceso o para la réplica. Los archivos del registro histórico aumentan y disminuyen de tamaño para acomodarse a los registros que hay que almacenar. Se pueden añadir más archi-

vos del registro histórico a la base de datos (en nuevos discos, por ejemplo) mientras que el sistema se está ejecutando y sin bloquear ninguna operación actual, y todos los registros históricos se tratan como si fueran un archivo continuo.

El sistema de recuperación de SQL Server tiene muchos aspectos en común con el algoritmo de recuperación ARIES (consulte la Sección 16.8), y en esta sección se muestran algunas de las diferencias fundamentales.

SQL Server cuenta con una opción de configuración denominada **intervalo de recuperación,** que permite que el administrador limite el tiempo que SQL Server debe tardar en recuperarse después de una caída. El servidor ajusta dinámicamente la frecuencia de los puntos de comprobación para reducir el tiempo de recuperación a valores comprendidos dentro del intervalo de recuperación. Los puntos de comprobación eliminan todas las páginas desfasadas de la memoria intermedia agrupada y se ajustan a las capacidades del sistema de E/S y a su carga de trabajo actual para eliminar de forma efectiva cualquier efecto sobre las transacciones que se estén ejecutando.

En un primer momento, después de una caída, el sistema inicia varias hebras (dimensionadas automáticamente al número de CPU) para comenzar la recuperación de varias bases de datos en paralelo. La primera fase de la recuperación es una pasada de análisis del registro histórico, que crea una tabla de páginas desfasadas y una lista de transacciones activas. La siguiente fase es una pasada de rehacer que se inicia desde el último punto de comprobación y rehace todas las operaciones. Durante la fase de rehacer se usa la tabla de páginas desfasadas para leer anticipadamente las páginas de datos. La fase final es una fase de deshacer en la que se retroceden las transacciones incompletas. La fase de deshacer se divide realmente en dos partes, puesto que SQL Server usa un esquema de recuperación de dos niveles. Las transacciones del primer nivel (aquellas que implican operaciones internas como la asignación de espacio y las divisiones de páginas) retroceden en primer lugar, seguidas por las transacciones de los usuarios. Una vez que las transacciones del primer nivel han retrocedido, la base de datos se conecta con el exterior y queda disponible para que comiencen nuevas transacciones de los usuarios mientras se llevan a cabo las últimas operaciones de retroceso. Esto se consigue haciendo que la pasada de rehacer vuelva a adquirir los bloqueos para todas las transacciones de usuario incompletas, que retrocederán en la fase de deshacer.

30.5.3.2. Recuperación de los medios

Las capacidades de copia de seguridad y de restauración de SQL Server permiten que se recupere de muchos fallos, incluidos la pérdida o corrupción de los medios de disco, los errores del usuario y la pérdida permanente de servidores. Además, la realización de copias de seguridad y de restauraciones de las bases de datos es útil para otros fines, como la copia de bases de datos de un servidor a otro y el mantenimiento de sistemas en espera.

SQL Server tiene tres modelos de recuperación diferentes entre los que los usuarios pueden elegir para cada base de datos. Mediante la especificación del modelo de recuperación, el administrador declara el tipo de capacidades de recuperación que se necesitan (como la restauración en un momento determinado y el envío de registros históricos) y las copias de seguridad necesarias para conseguirlas. Se pueden realizar copias de seguridad de las bases de datos, de los archivos, de los grupos de archivos y del registro histórico de transacciones. Todas las copias de seguridad son difusas y se realizan completamente en línea; es decir, no bloquean ninguna operación de LMD ni de LDD mientras se ejecutan. Las recuperaciones también se pueden llevar a cabo en línea, de modo que solo se deje desconectada la parte de la base de datos que se está recuperando (por ejemplo, un bloque de disco corrupto). Las operaciones de copia de seguridad y de restauración están muy optimizadas y solo quedan limitadas por la velocidad de los medios a los que se dirige la copia de seguridad. SQL Server puede realizar copias de seguridad tanto en dispositivos de disco como de cinta (hasta 64 en paralelo) y tiene API de gran rendimiento para usarlas con productos de copia de seguridad de otros fabricantes.

30.5.3.3. Copias espejo de las bases de datos

El uso de copias espejo de las bases de datos supone la reproducción inmediata de todas las actualizaciones de una de las bases de datos (la base de datos principal) en una copia diferente y completa de la base de datos (la base de datos copia espejo) que se suele ubicar en otra máquina. En caso de desastre en el servidor principal, o simplemente por labores de mantenimiento, el sistema puede recurrir de manera inmediata a la copia espejo en cuestión de segundos. La biblioteca de comunicación que se usa en las aplicaciones conoce la configuración espejo y, entonces, automáticamente reconecta a la máquina espejo en caso de que se produzca un fallo del sistema. Se consigue un estrecho acoplamiento entre la base de datos principal y la copia espejo mediante el envío de bloques de registros históricos de transacciones a la copia espejo a medida que se generan en la base de datos principal y mediante la reconstrucción de los registros del registro histórico en la copia espejo. En el modo de seguridad completa las transacciones no se pueden comprometer hasta que los registros del registro histórico de la transacción han llegado al disco de la copia espejo. Además de utilizarse en caso de error, la copia espejo se puede usar para restaurar automáticamente una página copiándola desde el espejo en caso de que se descubra que la página está corrupta durante un intento por leerla.

30.6. Arquitectura del sistema

Cada ejemplar de SQL Server es un único proceso del sistema operativo, que es también un punto de referencia para las solicitudes de ejecución de SQL. Las aplicaciones interactúan con SQL Server mediante diferentes bibliotecas cliente (como ODBC y OLE-DB) con el fin de ejecutar SQL.

30.6.1. Agrupación de hebras en el servidor

Para minimizar el cambio de contextos en el servidor y para controlar el grado de multiprogramación, el proceso de SQL Server mantiene un grupo de hebras que ejecutan las solicitudes del cliente. Cuando llegan las solicitudes del cliente se les asigna una hebra en la que ejecutarse. La hebra ejecuta las sentencias de SQL enviadas por el cliente y devuelve el resultado. Una vez completada la solicitud del usuario, la hebra es devuelta al grupo de hebras. Además de las solicitudes de los usuarios, el grupo de hebras también se usa para asignar hebras para tareas internas que se ejecutan en segundo plano, como:

- **Escritor diferido** (*lazywriter*). Esta hebra se dedica a garantizar que una cierta cantidad del grupo de memorias intermedias está libre y disponible en todo momento para su asignación por el sistema. Esta hebra también interactúa con el sistema operativo para determinar la cantidad óptima de memoria que debe consumir el proceso de SQL Server.
- **Punto de comprobación** (*checkpoint*). Esta hebra comprueba de forma periódica todas las bases de datos para mantener un intervalo de recuperación breve para el inicio de las bases de datos al reiniciar el servidor.
- **Monitor de interbloqueo** (*deadlock monitor*). Esta hebra supervisa otras hebras, buscando interbloqueos en el sistema. Es responsable de la detección de interbloqueos y también selecciona una víctima para permitir que el sistema progrese.

Cuando el procesador de consultas elige un plan paralelo para ejecutar una consulta determinada puede asignar varias hebras que trabajen en nombre de la hebra principal para ejecutar la consulta. Puesto que la familia de sistemas operativos de Windows NT proporciona soporte nativo de las hebras, SQL Server usa las hebras de NT para su ejecución. No obstante, SQL Server se puede configurar para que ejecute hebras en modo usuario además de las hebras del núcleo en sistemas de prestaciones muy elevadas para evitar el coste de un cambio de contexto del núcleo en los intercambios de hebras.

30.6.2. Gestión de la memoria

Hay muchos usos distintos de la memoria en el proceso de SQL Server:

- **Grupo de memorias intermedias.** El mayor consumidor de memoria del sistema es el grupo de memorias intermedias. El grupo de memorias intermedias mantiene una caché de las páginas de la base de datos usadas más recientemente. Usa un algoritmo de sustitución de reloj con una política de expropiación no forzosa; es decir, las páginas de la memoria intermedia con actualizaciones no comprometidas se pueden sustituir («expropiar»), y no se fuerza su envío al disco cuando se compromete la transacción. Las memorias intermedias también obedecen el protocolo del registro histórico de escritura anticipada para garantizar la corrección de la recuperación de las caídas y de los medios.
- **Asignación de la memoria dinámica.** Se trata de la memoria que se asigna de forma dinámica para ejecutar solicitudes remitidas por el usuario.
- **Caché de planes y de ejecución.** Esta caché almacena los planes compilados para varias consultas que los usuarios han ejecutado previamente en el sistema. Esto permite que varios usuarios compartan el mismo plan (lo cual ahorra memoria) y también ahorra tiempo de compilación de la consulta para consultas parecidas.
- **Concesiones de grandes cantidades de memoria.** Para los operadores de consulta que consumen grandes cantidades de memoria, como las reuniones por asociación y las ordenaciones.

SQL Server usa un esquema elaborado de gestión de la memoria para dividir su memoria entre los diversos usos que se han descrito. Un solo administrador de memoria gestiona de forma centralizada toda la memoria usada por SQL Server. El administrador de memoria es responsable de realizar de forma dinámica la división y la redistribución de la memoria entre los diversos consumidores de memoria del sistema. Distribuye esa memoria de acuerdo con un análisis de la relación entre costes y beneficios de la memoria para cualquier uso concreto. Hay disponible un mecanismo generalizado con infraestructura LRU para todos los componentes. Esta infraestructura de caché no solo realiza un seguimiento del tiempo de vida de los datos guardados en la caché, sino también de los costes relativos de CPU y de E/S necesarios para crearlos y guardarlos en la caché. Esta información se utiliza para determinar los costes relativos de los diferentes datos guardados en la caché. El administrador de memoria se centra en la expulsión de los datos guardados en la caché que no han sido utilizados recientemente y que eran baratos de guardar. Como ejemplo, es más probable que permanezcan en la memoria los planes de consulta complejos que necesitan segundos de CPU para compilarse que los planes triviales para frecuencias de acceso equivalentes.

El administrador de memoria interactúa con el sistema operativo para decidir de forma dinámica la cantidad de memoria que se debe consumir de la cantidad total de memoria del sistema. Esto permite que SQL Server sea bastante agresivo en el uso de la memoria del sistema, pero también que pueda devolver memoria al sistema cuando otros programas la necesiten sin causar fallos de página excesivos.

Además, el administrador de memoria conoce la topología de la CPU y la memoria del sistema. Concretamente, aprovecha el NUMA (acceso a memoria no uniforme) que muchos equipos emplean e intenta mantener el carácter local entre el procesador en que ejecuta una hebra y la memoria a la que accede.

30.6.3. Seguridad

SQL Server ofrece mecanismos de seguridad generales y directivas para la autenticación, la autorización, la auditoría y el cifrado. La autenticación puede realizarse mediante un par nombre de usuario-contraseña que gestione SQL Server o mediante una cuenta de Windows. La autorización se gestiona a través de concesiones de permisos para objetos de esquema o con permisos sobre objetos contenedores, como una base de datos o un servidor. En la comprobación de la autorización los permisos se calculan contabilizando el ámbito de los mismos y la pertenencia a un rol principal. Las auditorías se definen de la misma forma que los permisos: en objetos de esquema para un rol principal dado o sobre objetos contenedores; y en el momento de la operación se calculan dinámicamente según la definición de auditoría de los objetos y para cualquier tipo de auditoría o pertenencia a un rol principal. Se pueden definir distintas auditorías con propósitos diversos, como las de HIPAA[1] y Sarbanes-Oxley, y gestionar de forma independiente sin riesgo de que interfieran una con otra. Los registros de auditoría se escriben en un archivo o en el registro de seguridad de Windows.

SQL Server proporciona tanto cifrado manual de los datos como cifrado de datos transparente (Transparent Data Encryption). Este último cifra todas las páginas de datos y las registra cuando se escriben en el disco, descifrándolas cuando se leen del disco de forma que permanezcan cifradas en el disco pero en forma legible para los usuarios de SQL Server sin modificar la aplicación. Puede ser más eficiente en CPU que el cifrado manual de los datos, ya que los datos solo se cifran cuando se escriben en el disco y se realiza en grandes unidades, páginas, en lugar de con celdas individuales de datos.

Dos cosas son todavía más esenciales para la seguridad de los usuarios: (1) la calidad de todo el código base y (2) la posibilidad de que los usuarios determinen si han protegido adecuadamente el sistema. La calidad del código base se mejora usando el ciclo de vida de desarrollo de seguridad (Security Development Lifecycle). Los desarrolladores y probadores del producto reciben formación sobre seguridad y todas las características se modelan como amenazas para protegerlas apropiadamente. Siempre que es posible, SQL Server usa las características de seguridad subyacentes del sistema operativo en vez de implementar las suyas propias, como utilizar la autorización de Windows y el registro de seguridad de Windows para el registro de auditoría. Además, se usan numerosas herramientas internas para analizar el código base, en busca de posibles fallos de seguridad. La seguridad se verifica usando pruebas borrosas[2] y pruebas del modelo de amenazas. Antes de liberarlo, se hace una revisión final de seguridad del producto y un plan de respuesta para tratar los problemas de seguridad que se encuentren tras liberarlo, que es cuando se ejecuta y se descubren nuevos problemas.

1 La Ley de Sarbanes-Oxley es una ley de regulación financiera del gobierno estadounidense. La HIPAA es otra ley gubernamental relativa a la salud, que incluye regulaciones sobre la información relacionada con la salud.

2 Prueba borrosa: técnica basada en la aleatoriedad que analiza posibles entradas no válidas inesperadas.

Varias de estas características se proporcionan para ayudar a los usuarios a proteger el sistema adecuadamente. Una de ellas es una directiva fundamental denominada «deshabilitado de forma predeterminada», mediante la cual se deshabilitan de forma predeterminada muchos componentes usados escasamente de entre los que necesitan una preocupación adicional por la seguridad. Otra característica es un «analizador de buenas prácticas», que avisa a los usuarios sobre las configuraciones de los parámetros del sistema que pueden provocar vulnerabilidades de seguridad. La administración basada en políticas permite a los usuarios definir cuál será la configuración y avisar o prevenir respecto a cambios que pudiesen entrar en conflicto con la configuración aprobada.

30.7. Acceso a los datos

SQL Server soporta las siguientes interfaces de programación de aplicaciones (*application programming interface*: API) para la creación de aplicaciones intensivas de datos:

- **ODBC** (Conectividad abierta de bases de datos: Open Database Connectivity). Se trata de la implementación de Microsoft de la interfaz del nivel de llamadas (*call-level interface*: CLI) de la norma SQL:1999. Incluye los modelos de objetos Remote Data Objects, RDO (objetos de datos remotos) y Data Access Objects, DAO (objetos de acceso a datos), que facilitan la programación de las aplicaciones de bases de datos multicapa a partir de lenguajes de programación como Visual Basic.
- **OLE-DB** (Vinculación e incrustación de objetos para bases de datos: Object Linking and Embedding-Database). Se trata de una API de bajo nivel, orientada a sistemas, diseñada para los programadores que crean componentes de bases de datos. La interfaz está construida de acuerdo con el modelo de objetos componentes (Component Object Model: COM) de Microsoft, y permite la encapsulación de servicios de bajo nivel de la base de datos, como los proveedores de conjuntos de filas, los proveedores ISAM y los motores de consultas. OLE-DB se usa en SQL Server para integrar el procesador de consultas relacionales y el motor de almacenamiento, y permitir la réplica y el acceso distribuido a SQL y a otros orígenes externos de datos. Al igual que ODBC, OLE-DB incluye un modelo de objetos de nivel superior denominado Objetos de datos ActiveX (ActiveX Data Objects: ADO) para facilitar la programación de aplicaciones de bases de datos desde Visual Basic.
- **ADO.NET.** Se trata de una API diseñada para las aplicaciones escritas en lenguajes .NET, como C# y Visual Basic.NET. Esta interfaz simplifica algunos patrones frecuentes de acceso a datos soportados por ODBC y por OLE-DB. Además, proporciona un nuevo modelo de *conjuntos de datos* (*data set*) para permitir las aplicaciones de acceso a datos desconectadas y sin estado. ADO.NET incluye el **marco de entidad** de ADO.NET (Entity Framework), que es una plataforma para la programación contra datos que eleva el nivel de abstracción desde el nivel lógico (relacional) al nivel conceptual (entidad) y, por tanto, reduce significativamente la desconexión para los servicios de aplicaciones y datos como los informes, el análisis y la réplica. El modelo de datos conceptual se implementa utilizando un modelo relacional extendido, el **modelo dato entidad** (Entity Data Model: EDM), que aúna entidades y relaciones como conceptos de primera clase. Incluye un lenguaje de consulta para el EDM, denominado **SQL entidad** (SQL Entity), un motor de asociación que traduce desde el nivel conceptual al lógico (relacional) y un conjunto de herramientas dirigidas por el modelo que ayudan a los desarrolladores a definir asociaciones entre objetos y entidades a tablas.
- **LINQ. Lenguaje integrado de consulta,** o LINQ, permite utilizar directamente construcciones declarativas orientadas a conjuntos en lenguajes de programación como C# y Visual Basic. Las expresiones de consulta no se procesan con una herramienta o preprocesador externo sino que son expresiones de primera clase del propio lenguaje. LINQ permite que las expresiones de consultas se beneficien de los metadatos; la comprobación de la sintaxis en tiempo de compilación; el tipado estático y la autocompleción, que ya estaba disponible solo para el código imperativo. LINQ define un conjunto de operadores de consulta estándar de propósito general que permiten expresar operaciones de recorridos, filtrado, reunión, proyección, ordenación y agrupación de forma declarativa directa en cualquier lenguaje de programación de .NET. C# y Visual Basic también incluyen estas consultas, es decir, las extensiones de sintaxis del lenguaje que permiten operadores de consulta estándar.
- **DB-Lib.** La biblioteca de bases de datos para la API de C que se desarrolló específicamente para ser usada por versiones de SQL Server anteriores a la norma SQL-92.
- **HTTP/SOAP.** Las aplicaciones pueden usar las solicitudes HTTP/SOAP para invocar las consultas y los procedimientos de SQL Server. Las aplicaciones pueden usar URL que especifiquen raíces virtuales de Internet Information Server (Servidor de Información de Internet, IIS) que hagan referencia a ejemplares de SQL Server. La URL puede contener una consulta XPath, una instrucción de Transact-SQL o una plantilla de XML.

30.8. Procesamiento de consultas heterogéneas distribuidas

La posibilidad de realizar consultas distribuidas heterogéneas de SQL Server permite formular consultas transaccionales y actualizaciones de gran variedad de orígenes relacionales y no relacionales mediante proveedores de datos OLE-DB que se ejecutan en una o más computadoras. SQL Server soporta dos métodos para hacer referencia a los orígenes de datos OLE-DB heterogéneos en las instrucciones Transact-SQL. El método de los nombres de servidor vinculados utiliza procedimientos almacenados del sistema para asociar el nombre de un servidor con cada origen de datos OLE-DB. Se puede hacer referencia a los objetos de estos servidores vinculados en las instrucciones de Transact-SQL usando el convenio de nombres de cuatro partes que se describe más adelante. Por ejemplo, si el nombre de un servidor vinculado de *DeptSQLSrvr* se define en otra copia de SQL Server, la siguiente instrucción hace referencia a una tabla de ese servidor:

```
select *
from DeptSQLSrvr.Northwind.dbo.Employees;
```

En SQL Server los orígenes de datos OLE-DB se registran como servidores vinculados. Una vez que se define un servidor vinculado, se puede tener acceso a sus datos usando el nombre de cuatro partes:

```
<servidor_vinculado>.<catálogo>.<esquema>.<objeto>
```

El siguiente ejemplo establece un servidor vinculado a un servidor Oracle mediante un proveedor OLE- DB para Oracle:

```
exec sp addlinkedserver OraSvr, 'Oracle 7.3',
                        'MSDAORA', 'OracleServer'
```

Una consulta a este servidor vinculado se expresa como:

```
select *
from OraSvr.CORP.ADMIN.SALES;
```

Además, SQL Server soporta funciones intrínsecas parametrizadas de tipo tabla, denominadas **openrowset** y **openquery,** que permiten enviar consultas no interpretadas a proveedores o a servidores vinculados, respectivamente, en el dialecto soportado por cada proveedor. La siguiente consulta combina la información almacenada en un servidor de Oracle y en un servidor de Microsoft Index Server. Devuelve una lista de todos los documentos y sus autores que contengan las palabras Datos y Acceso ordenada por el departamento, así como por el nombre del autor.

```
select e.dept, f.AutorDoc, f.NombreArchivo
from OraSvr.Corp.Admin.Empleado e,
openquery(ArchivosEmp,
   'select AutorDoc, NombreArchivo
   from scope("c:\EmpDocs")
   where contains(' ' "Datos" near()"Acceso" ' ')>0') as f
where e.nombre = f.AutorDoc
order by e.dept, f.AutorDoc;
```

El motor relacional usa las interfaces OLE-DB para abrir los conjuntos de filas de los servidores vinculados, capturar las filas y gestionar las transacciones. Para cada origen de datos OLE-DB al que se tiene acceso como servidor vinculado debe estar presente un proveedor OLE-DB en el servidor en el que se ejecuta SQL Server. El conjunto de operaciones de Transact-SQL que se pueden usar en un origen de datos OLE-DB concreto depende de las capacidades del proveedor OLE-DB. Siempre que sea efectivo en cuanto al coste, SQL Server envía operaciones relacionales como reuniones, restricciones, proyecciones, ordenaciones y agrupaciones al origen de datos OLE-DB. SQL Server utiliza el coordinador de transacciones distribuidas de Microsoft (Microsoft Distributed Transaction Coordinator) y las interfaces de transacciones OLE-DB del proveedor para garantizar la atomicidad de las transacciones que abarcan varios orígenes de datos.

30.9. Réplica

La réplica de SQL Server la forman un conjunto de tecnologías para la copia y distribución de datos y de objetos de las bases de datos entre unas bases de datos y otras, el seguimiento de las modificaciones y la sincronización entre las bases de datos para conservar la consistencia. Las versiones más recientes de la duplicación de SQL Server también ofrecen la duplicación en línea de la mayor parte de las modificaciones de los esquemas de las bases de datos sin necesidad de interrupciones ni reconfiguraciones.

Los datos se suelen replicar para aumentar su disponibilidad. La replicación puede reunir datos corporativos de sitios geográficamente dispersos con destino a informes, distribuir datos a usuarios remotos de redes de área local o usuarios itinerantes de conexiones de acceso telefónico a redes o de Internet. La replicación de Microsoft SQL Server también mejora el rendimiento de las aplicaciones mediante su dimensionado con objeto de incrementar el rendimiento total de lectura entre duplicados, como es habitual al proporcionar servicios de caché de datos de capa intermedia para sitios web.

30.9.1. Modelo de replicación

SQL Server introdujo la metáfora **Publicar-Suscribir** para la replicación de las bases de datos y extiende esta metáfora de la industria editorial a sus herramientas de administración y supervisión de las réplicas.

El **publicador** es un servidor que pone los datos a disposición de otros servidores para su réplica. El publicador puede tener una o más publicaciones, cada una de las cuales representa un conjunto de datos y de objetos de la base de datos relacionados lógicamente. Los objetos discretos de cada publicación, incluidas las tablas, los procedimientos almacenados, las funciones definidas por el usuario, las vistas, las vistas materializadas, etc., se denominan **artículos.** Añadir un artículo a una publicación permite la personalización extensiva de la forma en que se replica ese objeto, por ejemplo, las restricciones sobre los usuarios que pueden suscribirse para recibir esos datos y la manera en que ese conjunto de datos debe filtrarse de acuerdo con la proyección o la selección de una tabla, mediante un filtro «horizontal» o «vertical», respectivamente.

Los **suscriptores** son servidores que reciben los datos replicados de los publicadores. Los suscriptores se pueden suscribir, como resulte oportuno, solo a las publicaciones que necesiten de uno o varios publicadores, independientemente del número o del tipo de opciones de replicación que implemente cada uno. Dependiendo del tipo de opciones de replicación seleccionadas, el suscriptor se puede usar como réplica de solo lectura o bien se pueden realizar modificaciones en los datos que se propagan automáticamente al publicador y, por consiguiente, al resto de réplicas. Los suscriptores también pueden volver a publicar los datos a los que se suscriben, dando soporte a una topología de replicación tan flexible como la empresa necesite.

El **distribuidor** es un servidor que desempeña varios roles, en función de las opciones de replicación escogidas. Como mínimo, se usa como repositorio de la información histórica y de los estados de error. En otros casos se utiliza también como cola intermedia de almacenamiento y entrega para redimensionar la entrega de la carga de replicación a todos los suscriptores.

30.9.2. Opciones de replicación

La replicación de Microsoft SQL Server ofrece un amplio espectro de opciones tecnológicas. Para decidir sobre las opciones de replicación adecuadas que se pueden usar, el diseñador de bases de datos debe determinar los requisitos de la aplicación con respecto a la operación autónoma del sitio implicado y el grado de consistencia transaccional necesario.

La **replicación instantánea** copia y distribuye los datos y los objetos de la base de datos exactamente como aparecen en un momento dado. La replicación instantánea no exige un seguimiento continuo de las modificaciones, puesto que los cambios no se propagan a los suscriptores de forma incremental. Los suscriptores se actualizan de manera periódica con una renovación completa del conjunto de datos definido por la publicación. Las opciones disponibles para la replicación instantánea pueden filtrar los datos publicados y permitir que los suscriptores modifiquen los datos replicados y propaguen esos cambios al publicador. Este tipo de replicación es más indicado para datos de pequeño tamaño y cuando las actualizaciones suelen afectar a datos suficientes como para que la replicación de una renovación completa de los datos resulte eficiente.

Con la **replicación transaccional** el publicador propaga una instantánea inicial de los datos a los suscriptores y luego envía las modificaciones incrementales de esos datos a los suscriptores en forma de transacciones discretas y comandos. El seguimiento del cambio incremental se produce dentro del motor principal de SQL Server, que marca las transacciones que afectan a los objetos replicados en el registro histórico de transacciones de la base de datos publicadora. Un proceso de replicación denominado **agente de lectura del registro** (histórico) lee estas transacciones del registro histórico de transacciones de la base de datos, aplica un filtro opcional y las almacena en la base de datos de distribución, que actúa como cola fiable soportando el mecanismo de almacenamiento y entrega de la replicación transaccional (el concepto de cola fiable es el mismo que el de colas duraderas descrito en la Sección 26.1.1). Otro proceso de replicación, denominado **agente de distribución,** envía posteriormente las modificaciones a cada suscriptor. Al igual

que la replicación instantánea, la replicación transaccional ofrece a los suscriptores la opción de realizar actualizaciones que utilicen un compromiso de dos fases que refleje esas modificaciones de forma consistente en el publicador o de colocarlas en cola en el suscriptor para su recuperación asíncrona mediante un proceso de replicación que propague posteriormente la modificación al publicador. Este tipo de replicación resulta adecuada cuando hay que conservar los estados intermedios entre varias actualizaciones.

La **replicación por mezcla** permite que cada réplica de la empresa funcione con total autonomía tanto en conexión como sin conexión. El sistema realiza un seguimiento de los metadatos según las modificaciones de los objetos publicados en los publicadores y en los suscriptores de todas las bases de datos replicadas, y el agente de replicación mezcla esas modificaciones de los datos durante la sincronización entre los pares replicados y asegura la convergencia de los datos mediante la detección y resolución automática de los conflictos. El agente de replicación usado en el proceso de sincronización incorpora numerosas opciones de políticas de resolución de conflictos, y se puede escribir una política de resolución de conflictos personalizada mediante el uso de procedimientos almacenados o de una interfaz extensible del **modelo de objetos componentes** (*component object model*: COM). Este tipo de replicación no replica todos los estados intermedios, sino solo el estado actual de los datos en el momento de la sincronización. Resulta adecuado cuando las réplicas necesitan la posibilidad de llevar a cabo actualizaciones autónomas mientras no se encuentran conectadas a ninguna red.

30.10. Programación de servidores en .NET

SQL Server soporta el motor común en tiempo de ejecución de lenguajes .NET (.NET Common Language Runtime, CLR) en el proceso de SQL Server para permitir que los programadores de bases de datos escriban la lógica corporativa en forma de funciones, procedimientos almacenados, disparadores, tipos de datos y agregados. La posibilidad de ejecutar el código de las aplicaciones dentro de la base de datos añade flexibilidad al diseño de las arquitecturas de las aplicaciones que necesita que la lógica corporativa se ejecute cerca de los datos y no puede permitirse el coste de enviar estos a un proceso de una capa intermedia para llevar a cabo el cálculo fuera de la base de datos.

CLR (.NET Common Language Runtime) es un entorno de tiempo de ejecución con un lenguaje intermedio fuertemente tipado que ejecuta varios lenguajes de programación modernos, como C#, Visual Basic, C++, COBOL y J++, entre otros, tiene memoria con recogida de basura, hebras con derecho preferente, servicios de metadatos (reflexión de tipos), comprobación de los tipos y seguridad de acceso al código. El motor en tiempo de ejecución usa los metadatos para localizar y cargar las clases, dejar los ejemplares en la memoria, resolver las invocaciones a los métodos, generar código nativo, hacer que se cumpla la seguridad y definir las fronteras de contexto en el tiempo de ejecución.

El código de las aplicaciones se despliega en la base de datos mediante **ensamblados,** que son las unidades de empaquetado, implantación y versiones del código de las aplicaciones en .NET. El despliegue del código de las aplicaciones en la base de datos proporciona una forma uniforme de administrar, realizar copias de seguridad de las aplicaciones de bases de datos completas y restaurarlas (el código y los datos). Una vez se ha registrado un ensamblado en la base de datos, los usuarios pueden exponer los puntos de entrada del ensamblado mediante las instrucciones de LDD de SQL, que pueden actuar como funciones escalares o de tablas, procedimientos, disparadores, tipos y agregados mediante el uso de contratos de extensión bien definidos, que se hacen cumplir durante la ejecución de esas sentencias de LDD. Los procedimientos almacenados, los disparadores y las funciones suelen necesitar ejecutar consultas y actualizaciones de SQL. Esto se consigue mediante un componente que implementa el API de acceso a los datos ADO.NET para su uso dentro del proceso de la base de datos.

30.10.1. Conceptos básicos de .NET

En el entorno .NET, los programadores escriben el código de los programas en lenguajes de programación de alto nivel que implementan clases que definen su estructura (por ejemplo, los campos o las propiedades de las clases) y sus métodos. Algunos de esos métodos pueden ser funciones estáticas. La compilación del programa genera un archivo, denominado *ensamblado*, que contiene el código compilado en el **lenguaje intermedio de Microsoft** (*Microsoft Intermediate Language*: MSIL) y un *manifiesto* que contiene todas las referencias a los ensamblados dependientes. El manifiesto es parte integral de cada ensamblado y permite que se describa a sí mismo. El manifiesto del ensamblado contiene los metadatos del ensamblado, que describen todas las estructuras, campos, propiedades, clases, relaciones de herencia, funciones y métodos definidos en el programa. El manifiesto establece la identidad del ensamblado, especifica los archivos que conforman su implementación, especifica los tipos y recursos que lo constituyen, divide en elementos las dependencias del momento de la compilación respecto de otros ensamblados y especifica el conjunto de permisos necesarios para que el ensamblado se ejecute correctamente. Esta información se usa en el momento de la ejecución para resolver las referencias, hacer que se cumpla la directiva de vinculación de versiones y validar la integridad de los ensamblados cargados. El entorno .NET soporta un mecanismo accesorio denominado *atributos personalizados* para anotar las clases, las propiedades, las funciones y los métodos con información adicional o con facetas que las aplicaciones pueden desear capturar en los metadatos. Todos los compiladores .NET consumen estas anotaciones sin interpretarlas y las guardan en los metadatos del ensamblado. Todas esas anotaciones se pueden examinar del mismo modo que se examina cualquier otro metadato, mediante un conjunto común de API de reflexión. El **código gestionado** hace referencia al MSIL ejecutado en CLR en lugar del ejecutado directamente por el sistema operativo. Las aplicaciones de código gestionado obtienen servicios en el momento de la ejecución de lenguajes comunes, como la recogida automática de basura, la comprobación de tipos durante la ejecución y el soporte de seguridad. Estos servicios ayudan a ofrecer un comportamiento de las aplicaciones de código gestionado uniforme e independiente de las plataformas y de los lenguajes. Durante la ejecución, un compilador **en tiempo de ejecución** (*just-in-time*: JIT) traduce el MSIL a código nativo (por ejemplo, código de Intel X86). Durante esa traducción el código debe pasar un proceso de comprobación que examina el MSIL y los metadatos para averiguar si el código se puede considerar de tipo seguro.

30.10.2. CLR en SQL

SQL Server y CLR son dos motores de ejecución diferentes con modelos internos diversos de hebras, programación y gestión de la memoria. SQL Server soporta un modelo de hebras cooperativo sin derecho preferente en el que las hebras del SGBD entregan voluntariamente la ejecución de manera periódica o cuando esperan por los bloqueos o en la cola de E/S, mientras que CLR soporta un modelo de hebras apropiativo. Si el código del usuario que se ejecuta en el SGBD puede llamar directamente a las primitivas de multienhebramiento (subprocesamiento en DB2) del sistema operativo (SO), no se integra bien con el programador de tareas de SQL Server y puede degradar la capacidad de redimensionamiento

del sistema. CLR no distingue entre la memoria virtual y la física, mientras que SQL Server gestiona directamente la memoria física y se le exige que utilice la memoria física dentro de unos límites configurables.

Los diferentes modelos de multienhebramiento, programación y gestión de la memoria suponen un reto de integración para los SGBDs que se redimensionan para soportar millares de sesiones de usuario concurrentes. SQL Server resuelve este reto pasando a ser el sistema operativo de CLR cuando se alberga en el proceso de SQL Server. CLR llama a las primitivas de bajo nivel implantadas por SQL Server para multienhebramiento, planificación, sincronización y gestión de la memoria (consulte la Figura 30.5). Este enfoque proporciona las ventajas de redimensionamiento y fiabilidad siguientes:

Multienhebramiento, programación y sincronización comunes. CLR llama a las API de SQL Server para crear hebras, tanto para ejecutar el código del usuario como para su propio uso interno, como las hebras del recogedor de basura y del destructor de clases. Para sincronizar varias hebras, CLR llama a los objetos de sincronización de SQL Server. Esto permite que el programador de SQL Server programe otras tareas mientras una hebra espera en un objeto de sincronización. Por ejemplo, cuando CLR inicia la recogida de basura, todas sus hebras esperan a que acabe la recogida. Dado que las hebras de CLR y los objetos de sincronización en los que esperan son conocidos por el programador de SQL Server, este puede programar hebras que estén ejecutando otras tareas de la base de datos que no impliquen a CLR. Además, esto permite que SQL Server detecte los interbloqueos que implican a los bloqueos adoptados por los objetos de sincronización de CLR y usar técnicas tradicionales para su eliminación. El planificador de SQL Server tiene la posibilidad de detectar y de detener las hebras que no hayan aportado resultados durante un periodo de tiempo significativo. La posibilidad de vincular las hebras de CLR con las hebras de SQL Server implica que el planificador de SQL Server pueda identificar las hebras fuera de control que se ejecutan en CLR y gestionar su prioridad, de modo que no consuman recursos significativos de la CPU y afecten, por tanto, al flujo del sistema. Esas hebras fuera de control se suspenden y se vuelven a poner en la cola. A los infractores reincidentes no se les permiten intervalos de tiempo que sean injustos para otros trabajadores que estén ejecutando instrucciones. Si un infractor toma cincuenta veces la cantidad permitida, es castigado durante cincuenta «rondas» antes de que se le vuelva a permitir ejecutar instrucciones, ya que el planificador no puede decidir si un cálculo largo está fuera de control o es legítimo.

Gestión de la memoria común. CLR llama a las primitivas de SQL Server para la asignación y la desasignación de memoria. Dado que la memoria usada por CLR cuenta para el uso total de memoria del sistema, SQL Server puede seguir dentro de los límites de memoria del sistema configurados y garantizar que CLR y SQL Server no compitan entre sí por memoria. Además, SQL Server puede rechazar las solicitudes de memoria de CLR mientras el sistema está restringido y pedir al CLR que reduzca su uso de la memoria cuando otras tareas la necesiten.

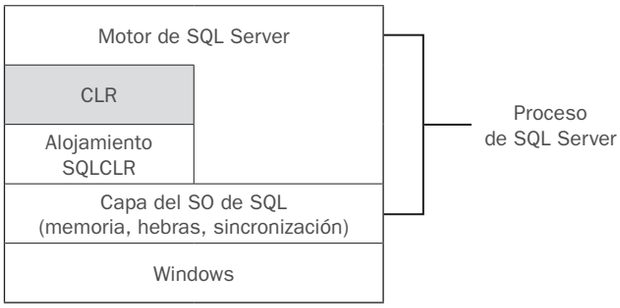


Figura 30.5. Integración de CLR con los servicios del sistema operativo de SQL Server.

30.10.3. Contratos para la extensión

Todo el código gestionado por el usuario que se ejecuta en el proceso de SQL Server interactúa con los componentes del SGBD como extensión. Entre las extensiones actuales están las funciones escalares, las tabulares, los procedimientos, los disparadores, los tipos escalares y los agregados escalares. Para cada extensión hay un contrato mutuo que define las propiedades o servicios que el código de usuario debe implementar para actuar como tal extensión, y los servicios que la extensión puede esperar del SGBD cuando se llame al código gestionado. CLR de SQL aprovecha la clase y la información de los atributos personalizados ya guardada en los metadatos del ensamblado para hacer que se cumpla que el código de usuario implemente esos contratos de extensión. Todos los ensamblados de usuario se guardan en la base de datos. Todos los metadatos relacionales y de los ensamblados se procesan en el motor de SQL mediante un conjunto uniforme de interfaces y de estructuras de datos. Cuando se procesan las instrucciones del lenguaje de definición de datos que registran una función, tipo o agregado de extensión dado, el sistema garantiza que el código de usuario implemente el contrato correspondiente mediante el análisis de los metadatos de su ensamblado. Si el contrato se implementa, la instrucción del LDD tiene éxito; en caso contrario, falla. Las subsecciones siguientes describen aspectos fundamentales de los contratos concretos que SQL Server hace cumplir actualmente.

30.10.3.1. Rutinas

Las funciones escalares, los procedimientos y los disparadores se clasifican genéricamente como rutinas. Las rutinas, implementadas como métodos estáticos de clase, pueden especificar las siguientes propiedades mediante los atributos personalizados.

- **IsPrecise.** Si esta propiedad binaria vale **false,** indica que el cuerpo de la rutina comprende cálculos imprecisos como las operaciones de coma flotante. Las expresiones que implican funciones imprecisas no se pueden indexar.
- **UserDataAccess.** Si el valor de esta propiedad es **read,** la rutina lee las tablas de datos de usuario. En caso contrario, el valor de la propiedad es *None*, lo que indica que la rutina no tiene acceso a los datos. Las consultas que no tienen acceso a las tablas de usuario (directamente o de manera indirecta mediante las vistas y las funciones) no se considera que tengan acceso a los datos de usuario.
- **SystemDataAccess.** Si el valor de esta propiedad es **read,** la rutina lee los catálogos del sistema o las tablas virtuales del sistema.
- **IsDeterministic.** Si esta propiedad vale **true,** se da por supuesto que la rutina produce el mismo resultado, dados los mismos valores de entrada, estado de la base de datos local y contexto de ejecución.
- **IsSystemVerified.** Este indica si SQL Server puede asegurarse de las propiedades de determinismo y de precisión o hacerlas cumplir (por ejemplo, funciones predefinidas y de Transact-SQL) o son tal y como las especifique el usuario (por ejemplo, funciones de CLR).
- **HasExternalAccess.** Si el valor de esta propiedad es **true,** la rutina tiene acceso a los recursos externos a SQL Server, como los archivos, la red, el acceso web y el registro.

30.10.3.2. Funciones que devuelven tablas

Las clases que implementan funciones que se valoran como tablas deben implementar la interfaz IEnumerable para permitir la iteración sobre las filas devueltas por la función, un método para describir el esquema de la tabla devuelta (es decir, las columnas, los tipos), un método para describir las columnas que pueden ser claves únicas y un método para insertar filas en la tabla.

30.10.3.3. Tipos

Las clases que implementan tipos definidos por los usuarios se anotan con el atributo SqlUserDefinedType(), que especifica las siguientes propiedades:

- **Format.** SQL Server soporta tres formatos de almacenamiento: nativo, definido por el usuario y serialización .NET.
- **MaxByteSize.** Se trata del tamaño máximo de la representación binaria serializada de los ejemplares de los tipos en bytes. UDT puede tener hasta 2GB.
- **IsFixedLength.** Se trata de una propiedad booleana que especifica si los ejemplares del tipo tienen longitud fija o variable.
- **IsByteOrdered.** Se trata de una propiedad booleana que indica si la representación binaria serializada de los ejemplares del tipo tiene ordenación binaria. Cuando esta propiedad vale **true,** el sistema puede realizar directamente comparaciones con esta representación sin necesidad de hacer que los ejemplares de los tipos sean objetos.
- **Nullability.** Todos los UDT del sistema deben ser capaces de guardar el valor NULL mediante el soporte de la interfaz INullable que contiene el método Boolean IsNull.
- **Type conversions.** Todos los UDT deben implementar las conversiones de tipo directas e inversas con las cadenas de caracteres mediante los métodos ToString y Parse.

30.10.3.4. Agregados

Además de soportar los contratos de los tipos, los agregados definidos por los usuarios deben implementar cuatro métodos exigidos por el motor de ejecución de consultas para inicializar el cálculo de los ejemplares agregados, para acumular valores de entrada en la función proporcionada por el agregado, para mezclar cálculos parciales del agregado y para recuperar el resultado final del agregado. Los agregados pueden declarar propiedades adicionales mediante los atributos personalizados en su definición de clase; esas propiedades las usa el optimizador de consultas para obtener planes alternativos para el cálculo del agregado.

- **IsInvariantToDuplicates.** Si esta propiedad vale **true,** el cálculo que entrega los datos al agregado puede modificarse descartando o introduciendo nuevas operaciones para la eliminación de duplicados.
- **IsInvariantToNulls.** Si esta propiedad vale **true,** se pueden descartar las filas NULL de la entrada. Sin embargo, hay que tener cuidado en el contexto de operaciones **group by** para no descartar grupos enteros.
- **IsInvariantToOrder.** Si esta propiedad vale **true,** el procesador de consultas puede ignorar las cláusulas **order by** y explorar los planes que eviten tener que ordenar los datos.

30.11. Soporte de XML

En los últimos años los sistemas de bases de datos relacionales han adoptado XML de muchas maneras diferentes. La primera generación del soporte de XML en los sistemas de bases de datos relacionales estaba relacionada sobre todo con la exportación de datos relacionales en forma de XML («publicación XML») y con la importación de datos relacionales en forma de marcas de XML de nuevo a una representación relacional («fragmentación de XML»). La situación en que más se usan estos sistemas es el intercambio de información en contextos en los que XML se utiliza como «formato cable» y los esquemas relacional y de XML suelen estar predefinidos de manera independiente entre sí. Para abordar esta situación, SQL Server ofrece amplia funcionalidad, como el agregador de publicación **for xml,** el proveedor de conjuntos de filas OpenXML y la tecnología de vistas de XML basada en los esquemas anotados.

La fragmentación de los datos XML en esquemas relacionales puede resultar bastante difícil o ineficiente para el almacenamiento de los datos semiestructurados cuya estructura puede variar con el tiempo, así como para almacenar documentos. Para soportar esas aplicaciones, SQL Server 2005 añade el soporte nativo de XML basado en el tipo de datos **xml** de SQL:2003. La Figura 30.6 ofrece un diagrama arquitectónico de alto nivel del soporte nativo de XML en las bases de datos de SQL Server. Consiste en la posibilidad de almacenar de manera nativa XML, restringir y tipificar los datos de XML almacenados con conjuntos de esquemas de XML y consultar y actualizar los datos de XML. Para proporcionar ejecuciones de consultas eficientes se proporcionan varios tipos de índices específicos de XML. Finalmente, el soporte nativo de XML también se integra con la «fragmentación» y la «publicación» directas e inversas de los datos relacionales.

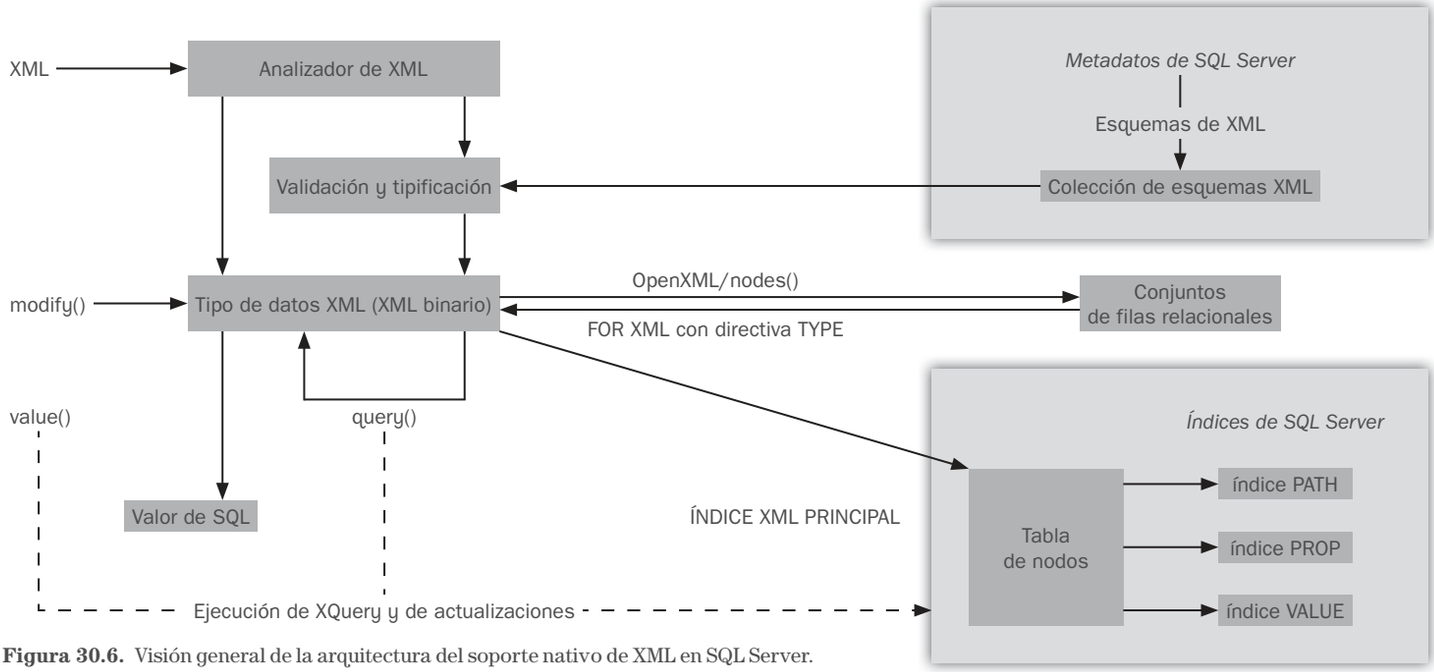


Figura 30.6. Visión general de la arquitectura del soporte nativo de XML en SQL Server.

30.11.1. Almacenamiento y organización nativos de XML

El tipo de datos **xml** puede guardar documentos de XML y contener fragmentos de contenido (varios nodos de texto o de elementos en la parte superior) y se define en términos del modelo de datos XQuery 1.0/XPath 2.0. Este tipo de datos se puede usar para parámetros de procedimientos almacenados, para variables y como tipo de columna.

SQL Server guarda los datos de tipo **xml** en un formato binario interno como **blob** y aporta mecanismos de indexado para la ejecución de consultas. El formato binario interno proporciona una recuperación y una reconstrucción eficientes del documento de XML original, además de algún ahorro de espacio (en promedio, un veinte por ciento). Los índices soportan un mecanismo eficiente de consulta que puede usar el motor y el optimizador de consultas relacionales; se ofrecen detalles más adelante, en la Sección 30.11.3.

SQL Server proporciona un concepto de metadato de base de datos llamado **colección de esquema XML** que asocia un identificador de SQL con una colección de componentes esquema de uno o más espacios de nombre objetivos.

30.11.2. Consultas y actualizaciones sobre el tipo de datos XML

SQL Server ofrece varias posibilidades de consultas y de modificación basadas en XQuery sobre el tipo de datos XML. Estas posibilidades de consulta y de modificación se soportan mediante los métodos definidos para el tipo de datos **xml.** Algunos de estos métodos se describen en el resto de esta sección.

Cada método toma el valor literal de una cadena de caracteres como cadena de caracteres de la consulta y, potencialmente, otros argumentos. El tipo de datos XML (al que se aplica el método) ofrece el elemento de contexto para las expresiones de ruta y rellena las definiciones de esquemas correspondientes al ámbito con toda la información de tipos proporcionada por el conjunto de esquemas de XML asociado (si no se proporciona ningún conjunto, se da por supuesto que los datos de XML no tienen ningún tipo). La implementación de XQuery de SQL Server tiene tipos estáticos, por lo que soporta la detección temprana de los errores de escritura de las expresiones de ruta, de los errores de tipos y de la no coincidencia de cardinalidades, así como algunas optimizaciones adicionales.

El método *query* toma expresiones de XQuery y devuelve ejemplares de tipos de datos de XML sin tipo (que luego se pueden convertir a un conjunto de esquemas objetivo si hace falta dar tipo a los datos). En la terminología de la especificación de XQuery se ha definido el modo de creación como «strip». El ejemplo siguiente muestra una expresión simple de XQuery que resume un elemento Cliente complejo de un documento de informe de viaje que contiene, entre otras informaciones, un nombre, un atributo ID y posibilidades de ventas que se contienen en las notas marcadas del informe de viaje real. El resumen muestra el nombre y la información sobre ventas de los elementos Cliente que tienen posibilidades de ventas.

```
select Informe.query ('
  declare namespace c = "urn:ejemplo/cliente";
    for $cust in /c:doc/c:cliente
    where $cust/c:notas//c:posiblesventas
    return
      <id_cliente id="$cli/@id"> {
        $cli/c:nombre,
        $cli/c:notas//c:posiblesventas
      }</cliente>')
from InformesViajes;
```

La consulta XQuery anterior se ejecuta para el valor XML guardado en el atributo *doc* de cada fila de la tabla *InformesViajes*. Cada una de las filas en el resultado de la consulta de SQL contiene el resultado de ejecutar la consulta de XQuery con los datos de una fila como entrada.

El método *value* toma una expresión de XQuery y el nombre de un tipo de SQL, extrae un solo valor atómico del resultado de la expresión de XQuery y convierte su forma léxica al tipo de SQL especificado. Si la expresión de XQuery da lugar a un nodo, el valor con tipo del nodo se extrae de manera implícita como valor atómico que se convierte al tipo de SQL (en la terminología de XQuery, el nodo se «atomiza»; el resultado se envía a SQL). Tenga en cuenta que el método *value* lleva a cabo una comprobación estática de tipos cerciorándose que se devuelve, como máximo, un valor.

El método *exist* toma una expresión de XQuery y devuelve un 1 si la expresión genera un resultado no vacío y un cero en caso contrario.

Finalmente, el método *modify* proporciona un mecanismo para modificar el valor de XML en el nivel de subárboles, insertando subárboles nuevos en ubicaciones concretas de cada árbol, modificando el valor de un elemento de un atributo y borrando subárboles. El ejemplo siguiente borra todos los elementos posiblesventas de los años anteriores al año proporcionado por una variable o por un parámetro de SQL de nombre @año:

```
update InformesViajes
set Informe.modify(
  'declare namespace c = "urn:ejemplo/cliente";
  delete /c:doc/c:cliente//c:posiblesventas
                          [@año < sql:variable("@año")]');
```

30.11.3. Ejecución de expresiones XQuery

Como ya se ha mencionado antes, los datos XML se guardan en una representación binaria interna. Sin embargo, para poder ejecutar las expresiones de XQuery, el tipo de datos XML se transforma internamente en una tabla de nodos. La tabla de nodos interna usa básicamente una fila para representar cada nodo. Cada nodo recibe un identificador OrdPath como identificador de nodo (los identificadores OrdPath son esquemas numéricos decimales Dewey modificados; consulte las notas bibliográficas para obtener referencias de más información sobre OrdPath). Cada nodo contiene también información de las claves para volver a apuntar a la fila de SQL original a la que pertenece el nodo, información sobre el nombre y el tipo (en forma simbólica), los valores, etc. Dado que OrdPath codifica tanto el orden de los documentos como la información de las jerarquías, la tabla de nodos se agrupa en términos de la información de las claves y de OrdPath, de modo que se pueda conseguir una expresión de ruta o la recomposición de un subárbol con la mera exploración de una tabla.

Todas las expresiones de XQuery y de actualización se traducen luego en un árbol de operadores algebraicos con la tabla de nodos interna; el árbol usa los operadores relacionales habituales y algunos operadores diseñados de manera específica para usar algebraicamente XQuery. El árbol resultante se injerta en el árbol algebraico de la expresión relacional de modo que, al final, el motor de ejecución de consultas recibe un solo árbol de ejecución que puede optimizar y ejecutar. Para evitar las costosas transformaciones en el momento de la ejecución los usuarios pueden materializar anticipadamente la tabla de nodos usando el índice principal de XML. SQL Server, además, proporciona tres índices secundarios de XML, de modo que la ejecución de las consultas pueda aprovecharse más aún de las estructuras de índices:

- El índice *path* (ruta) ofrece soporte para los tipos sencillos de expresiones de ruta.
- El índice *properties* (propiedades) ofrece soporte para la situación frecuente de comparaciones entre valores de las propiedades.
- El índice *value* (valor) es idóneo si la consulta usa comodines en las comparaciones.

Consulte las notas bibliográficas para hallar referencias a más información sobre el indexado de XML y el procesado de consultas de XML en SQL Server.

30.12. Service Broker de SQLServer

Service Broker ayuda a los desarrolladores a crear aplicaciones distribuidas de acoplamiento laxo al proporcionar soporte para la mensajería encolada y de confianza en SQL Server. Muchas aplicaciones de bases de datos usan el procesamiento asíncrono para mejorar la posibilidad de redimensionamiento y los tiempos de respuesta de las sesiones interactivas. Un enfoque habitual del procesamiento asíncrono es el uso de tablas de trabajos. En lugar de llevar a cabo todo el trabajo de un proceso corporativo en una sola transacción de la base de datos, la aplicación realiza una modificación que indica que ha aparecido un trabajo y luego inserta un registro del trabajo que hay que llevar a cabo en una tabla de trabajos. Cuando los recursos disponibles lo permiten, las aplicaciones procesan la tabla de trabajos y completan el proceso corporativo. Service Broker forma parte del servidor de bases de datos que soporta directamente este enfoque del desarrollo de aplicaciones. El lenguaje Transact-SQL incluye instrucciones de LDD y de LMD para Service Broker. Además, también se dispone de SQL Server Management Objects (SMO) para Service Broker en SQL Server. Este permite un acceso pragmático a los objetos de Service Broker desde el código administrado.

Las antiguas tecnologías de encolado de mensajes se concentraban en cada uno de los mensajes. Con Service Broker la unidad básica de comunicación es la **conversación:** un flujo de mensajes persistente, fiable a dos bandas. SQL Server garantiza que los mensajes de cada conversación se entregan a la aplicación exactamente una vez y en su orden. También es posible asignar una prioridad del 1 al 10 a una conversación. Los mensajes de conversaciones con mayor prioridad se envían y reciben más deprisa que los mensajes de conversaciones con menor prioridad. Las conversaciones tienen lugar entre dos servicios. Cada *servicio* es un punto final con nombre de una conversación. Cada conversación forma parte de un *grupo de conversaciones*. Las conversaciones relacionadas se pueden asociar al mismo grupo de conversaciones.

Las conversaciones y los mensajes están fuertemente tipificados; cada mensaje tiene un tipo concreto. SQL Server puede, opcionalmente, validar que los mensajes constituyan XML bien formado, que los mensajes estén vacíos o que un mensaje dado esté conforme con un esquema de XML. Un *contrato* define los tipos de mensaje admisibles para cada conversación y los participantes en esa conversación que pueden enviar mensajes de ese tipo. SQL Server ofrece un contrato y un tipo de mensajes predeterminados para las aplicaciones que solo necesitan un flujo digno de confianza.

SQL Server guarda los mensajes en tablas internas. Estas tablas no son accesibles directamente; en vez de esto, SQL Server muestra las *colas* como vistas de esas tablas internas. Las aplicaciones reciben mensajes de las colas. La operación receive (recibir) devuelve uno o varios mensajes del mismo grupo de conversaciones. Al controlar el acceso a la tabla subyacente, SQL Server puede hacer que se cumplan de manera eficiente la ordenación de los mensajes, la correlación de los mensajes relacionados y los bloqueos. Como las colas son tablas internas, no necesitan ningún tratamiento especial para las copias de seguridad, las restauraciones, los relevos ni las copias exactas de las bases de datos. Tanto las tablas de las aplicaciones como los mensajes encolados asociados se incluyen en las copias de seguridad, se restauran y se relevan con la base de datos. Las conversaciones de Broker que se producen en las bases de datos con copias exactas continúan cuando parten al completarse el relevo de la copia exacta; aunque la conversación se produjera entre dos servicios ubicados en bases de datos diferentes.

La granularidad de los bloqueos de las operaciones de Service Broker es el grupo de conversaciones, en vez de una conversación concreta o los diferentes mensajes. Al forzar los bloqueos en los grupos de conversación, Service Broker ayuda de manera automática a que las aplicaciones eviten problemas de concurrencia al procesar los mensajes. Cuando una cola contiene varias conversaciones, SQL Server garantiza que solo un lector de colas pueda procesar los mensajes que pertenecen a un grupo de conversación dado a la vez. Esto elimina la necesidad de que las propias aplicaciones incluyan lógica para evitar los interbloqueos; una fuente de errores habitual en muchas aplicaciones de mensajería. Otro efecto lateral positivo de esta semántica de bloqueos es que las aplicaciones pueden decidir usar el grupo de conversaciones como clave para guardar y recuperar el estado de las aplicaciones. Estas ventajas del modelo de programación no son más que dos ejemplos de los beneficios que se obtienen de la decisión de formalizar la conversación como primitiva de la comunicación en lugar de la primitiva de mensajes atómicos usada en los sistemas tradicionales de encolado de mensajes.

SQL Server puede activar de manera automática los procedimientos almacenados cuando una cola contiene mensajes que hay que procesar. Para adecuar el número de procesos almacenados en ejecución al tráfico entrante, la lógica de activación supervisa la cola para ver si hay trabajo útil para otro lector de colas. SQL Server considera tanto la velocidad a la que los lectores existentes reciben los mensajes como el número de grupos de conversaciones disponibles para decidir si debe iniciar otro lector de colas. El procedimiento almacenado que se debe activar, el contexto de seguridad del procedimiento almacenado y el número máximo de ejemplares que se pueden iniciar se configuran para una cola dada. SQL Server también proporciona un Activador externo (External Activator). Esta característica permite activar una aplicación de fuera del SQL Server cuando se inserte un mensaje en una cola. La aplicación puede, entonces, recibir y procesar los mensajes. De esta forma, el trabajo intensivo de CPU se puede trasladar fuera del SQL Server a una aplicación, posiblemente en una computadora distinta. Así mismo, las tareas de larga duración, por ejemplo invocar un servicio web, se pueden ejecutar sin consumir recursos de la base de datos. El Activador externo sigue la misma lógica que la activación interna, y se puede configurar para activar varios ejemplares de una aplicación cuando los mensajes se acumulen en una cola.

Como extensión lógica de la mensajería asíncrona dentro del ejemplar, Service Broker también ofrece mensajería confiable entre ejemplares de SQL Server para permitir que los desarrolladores creen aplicaciones distribuidas con facilidad. Las conversaciones pueden tener lugar dentro de un solo ejemplar de SQL Server o entre dos ejemplares. Las conversaciones locales y las remotas usan el mismo modelo de programación.

La seguridad y el encaminamiento se configuran de manera declarativa, sin necesidad de modificaciones de los lectores de colas. SQL Server usa *rutas* para asignar un nombre de servicio a la

dirección de red del otro participante en la conversación y puede también llevar a cabo entregas de mensajes y equilibrados de carga sencillos para las conversaciones. SQL Server ofrece entregas confiables, solo una vez y en su orden, independientemente del número de ejemplares por los que tengan que viajar los mensajes. Las conversaciones que abarcan varios ejemplares de SQL Server se pueden proteger tanto en el nivel de la red (punto a punto) como en el nivel de la conversación (extremo a extremo). Cuando se usa la seguridad de extremo a extremo, el contenido de los mensajes permanece cifrado hasta que llega a su destino final, mientras que las cabeceras están disponibles para cada ejemplar de SQL Server por el que viaje cada mensaje. Los permisos estándar de SQL Server se aplican dentro de cada ejemplar. El cifrado se produce cuando los mensajes dejan un ejemplar.

SQL Server usa un protocolo binario para el envío de mensajes entre ejemplares. El protocolo fragmenta los mensajes de gran tamaño y permite que se intercalen los fragmentos de varios mensajes. La fragmentación permite que SQL Server transmita rápidamente los mensajes más pequeños, aun en el caso de que se esté transmitiendo un mensaje de gran tamaño. El protocolo binario no usa las transacciones distribuidas ni el compromiso de dos fases. En vez de eso, exige que un receptor reconozca los fragmentos de los mensajes. SQL Server se limita a volver a intentar enviar periódicamente los fragmentos de los mensajes hasta que el receptor los reconoce. Los reconocimientos se incluyen muy frecuentemente como parte de la cabecera de los mensajes devueltos, aunque se utilicen mensajes de vuelta exclusivos si no se dispone de ningún otro.

SQL Server incluye una herramienta de diagnóstico en línea de comandos (*ssbdiagnose*) para ayudar a analizar un despliegue de Service Broker e investigar los problemas. La herramienta puede ejecutarse en modo *configuración* o *en ejecución*. En el modo configuración, la herramienta comprueba si un par de servicios puede intercambiar mensajes y devuelve cualquier error de configuración. Entre los ejemplos de estos errores están las colas desactivadas y las rutas de vuelta desaparecidas. En el segundo modo, la herramienta conecta dos o más ejemplares de Service Broker y monitoriza los eventos de SQL Profiler para descubrir problemas de Service Broker en tiempo de ejecución. Los resultados de la herramienta se pueden enviar a un archivo para su procesamiento automático.

30.13. Inteligencia de negocio

El componente de almacenamiento de datos e inteligencia de negocio de SQL Server contiene tres subcomponentes:

- Los servicios de integración de SQL Server (*SQL Server Integration Services*: SSIS), que ofrecen los medios para integrar datos de varios orígenes, llevan a cabo transformaciones relacionadas con la limpieza de los datos y su transformación a un formato común y la carga de los datos en el sistema de bases de datos.
- Los servicios de análisis de SQL Server (*SQL Server Analysis Services*: SSAS), que proporcionan las capacidades OLAP y de minería de datos.
- Los servicios de informes de SQL Server (*SQL Server Reporting Services*: SSRS).

Los servicios de integración, los de análisis y los de informes se implementan en servidores diferentes y se pueden instalar de manera independiente en la misma máquina o en máquinas diferentes. Pueden conectar con gran variedad de orígenes de datos, como los archivos planos, las hojas de cálculo o gran variedad de sistemas de bases de datos relacionales, mediante conectores nativos, OLE-DB o controladores ODBC.

En conjunto, ofrecen una solución integrada para la extracción, transformación y carga de los datos, el modelado y la adición de capacidad analítica posteriores a los datos y, finalmente, la creación y distribución de informes de los datos. Los diversos componentes de la inteligencia de negocio de SQL Server pueden integrarse entre sí y aprovechar las capacidades de los demás. A continuación se muestran unas cuantas situaciones que aprovechan diferentes combinaciones de componentes:

- Creación de un paquete SSIS que limpie los datos, usando patrones generados por la minería de datos de SSAS.
- El uso de SSIS para cargar datos en un cubo de SSAS, su procesamiento y la ejecución de informes relativos al cubo de SSAS.
- Creación de un informe de SSRS para publicar los hallazgos de un modelo de minería o los datos contenidos en un componente OLAP de SSAS.

Las secciones siguientes aportan una visión general de las posibilidades y de la arquitectura de estos componentes de los servidores.

30.13.1. SQL Server Integration Services

SQL Server 2005 Integration Services (SSIS) de Microsoft es una solución corporativa para la transformación e integración de datos que se puede usar para extraer, transformar, agregar y consolidar datos de diferentes orígenes y trasladarlos a uno o varios destinos. Se puede usar SSIS para llevar a cabo las siguientes tareas:

- Mezclar datos de almacenes de datos heterogéneos.
- Refrescar los datos de los almacenes y los puestos de datos.
- Limpiar los datos antes de cargarlos en sus destinos.
- Realizar cargas masivas de datos en bases de datos de procesamiento de transacciones en línea (*Online Transaction Processing*: OLTP) y de procesamiento analítico en línea (*Online Analytical Processing*: OLAP).
- Enviar notificaciones.
- Incluir inteligencia de negocio en el proceso de transformación de los datos.
- Automatizar las funciones administrativas.

SSIS ofrece un conjunto completo de servicios, herramientas gráficas, objetos programables y API para estas tareas. Todo ello ofrece la posibilidad de crear soluciones de transformación de los datos de gran tamaño, robustas y complejas, sin necesidad de realizar una programación personalizada. No obstante, se dispone de API y de objetos programables cuando son necesarios para crear elementos personalizados o para integrar las posibilidades de transformación de los datos en aplicaciones personalizadas.

El motor de flujo de datos de SSIS ofrece las memorias intermedias ubicadas en la memoria que trasladan los datos desde su origen hasta su destino y llama a los adaptadores de orígenes que extraen los datos de los archivos y de las bases de datos relacionales. El motor también proporciona las transformaciones que modifican los datos y los adaptadores de destinos que cargan los datos en los almacenes de datos. La eliminación de duplicados basada en la coincidencia difusa (aproximada) es un ejemplo de la transformación proporcionada por SSIS. Los usuarios pueden programar sus propias transformaciones si fuese necesario. La Figura 30.7 muestra un ejemplo de combinación de varias transformaciones para la limpieza y carga de información sobre ventas de libros; los títulos de los libros de los datos de venta se comparan con la base de datos de publicaciones y, en caso de que no coincidan, se lleva a cabo una búsqueda difusa para manejar los títulos con errores sin importancia (como los de escritura). La información sobre la confianza y la correlación de los datos se guarda con los datos limpios.

30.13.2. SQL Server Analysis Services

El componente de servicios de análisis proporciona la funcionalidad de procesamiento analítico en línea (*On-Line Analytical Processing*: OLAP) y de minería de datos para las aplicaciones de inteligencia de negocio. Los servicios de análisis soportan una arquitectura de cliente ligero. El motor de cálculo se halla en el servidor, por lo que las consultas se resuelven allí, lo que evita la necesidad de transferir grandes cantidades de datos entre el cliente y el servidor.

30.13.2.1. SQL Server Analysis Services: OLAP

Los servicios de análisis (Analysis Services) utilizan el modelo dimensional unificado (*Unified Dimensional Model*: UDM), que cubre el hueco entre los informes relacionales tradicionales y el análisis OLAP ad hoc. El papel del modelo dimensional unificado es ofrecer un puente entre el usuario y los orígenes de datos. Los UDM se crean sobre uno o varios orígenes físicos de datos y luego el usuario final formula consultas al UDM, usando gran variedad de herramientas de cliente, como Microsoft Excel.

Además de ser una capa de modelado dimensional de los esquemas DataSource, el UDM proporciona un rico entorno para la definición de lógica, reglas y definiciones semánticas corporativas potentes y exhaustivas. Los usuarios pueden examinar y generar informes de los datos del UDM en su idioma nativo (por ejemplo, francés o hindi) mediante la definición de la traducción al lenguaje local del catálogo de metadatos y de los datos dimensionales. El servidor de análisis define dimensiones temporales complejas (fiscal, informes, fabricación, etc.) y permite la definición de lógica corporativa multidimensional potente (crecimiento interanual, en el año en curso) mediante el lenguaje de expresiones multidimensionales (MDX). El UDM permite que los usuarios definan perspectivas orientadas al negocio, cada una de las cuales solo presenta el subconjunto concreto del modelo (medidas, dimensiones, atributos, reglas corporativas, etc.) que es relevante para un grupo determinado de usuarios. Las empresas suelen definir indicadores de rendimiento cruciales (*key performance indicators*: KPI), que son métricas importantes que se utilizan para medir la prosperidad del negocio. Ejemplos de esos KPI son las ventas, los ingresos por empleado y la tasa de retención de los clientes. El UDM permite definir esos KPI, lo que hace posible una agrupación y presentación de los datos de forma mucho más comprensible.

30.13.2.2. SQL Server Analysis Services: minería de datos

SQL Server proporciona gran variedad de técnicas de minería, con una rica interfaz gráfica para ver los resultados de la minería. Algunos de los algoritmos soportados son:

- Las reglas de asociación (útiles para las aplicaciones de ventas cruzadas).
- Las técnicas de clasificación y predicción, como los árboles de decisión, los árboles de regresión, las redes neuronales y la lógica de Bayes ingenua.
- La predicción mediante series temporales, incluyendo ARIMA y ARTXP.
- Las técnicas de agrupación como la maximización de esperanzas y las medias- k (acopladas con técnicas de agrupación de secuencias).

Además, SQL Server proporciona una arquitectura extensible para la inclusión de algoritmos y visualizadores de minería de datos de terceros.

SQL Server también soporta las *Extensiones de minería de datos* (*Data-Mining Extensions*: DMX). DMX es el lenguaje usado para interactuar con los modelos de minería de datos, igual que SQL se usa para interactuar con las tablas y con las vistas. Con DMX se pueden crear, entrenar y almacenar modelos en bases de datos de los servicios de análisis. El modelo se puede examinar luego para buscar patrones o, mediante una sintaxis especial de **reunión de predicciones** (*prediction join*), aplicada a los datos nuevos, llevar a cabo predicciones. El lenguaje DMX soporta funciones y estructuras para determinar con facilidad las clases predichas, junto con su confianza; predecir una lista de elementos asociados, como en los motores de recomendaciones, o, incluso, devolver información y hechos que apoyen las predicciones. La minería de datos en SQL Server se puede usar con datos guardados en bases de datos relacionales o en orígenes de datos multidimensionales. Se soportan también otros orígenes de datos mediante tareas y transformaciones especializadas, lo que permite la minería de datos directamente en el cauce de datos operativo de los servicios de integración. Los resultados de la minería de datos se pueden presentar en controles gráficos, dimensiones especiales de minería de datos de los cubos OLAP o, simplemente, en los informes de los servicios de informes.

30.13.3. SQL Server Reporting Services

Los servicios de informes (Reporting Services) son una plataforma de informes basada en servidor que se puede usar para crear y gestionar informes tabulares, matriciales, gráficos y de formato libre que contengan datos de orígenes de datos relacionales y multidimensionales. Los informes creados se pueden ver y gestionar mediante conexiones basadas en web. Los informes matriciales pueden resumir datos de revisiones de alto nivel, mientras que ofrecen detalles de apoyo en los informes de minería. Los informes parametrizados se pueden utilizar para filtrar datos en términos de los valores que se proporcionen en el momento de la ejecución. Los usuarios pueden escoger entre gran variedad de formatos de vistas para presentar los informes sobre la marcha en los formatos preferidos para el tratamiento de datos o para la impresión. También se dispone de una API para extender o integrar capacidades de elaboración de informes en soluciones personalizadas. Los informes basados en los servidores ofrecen una manera de centralizar el almacenamiento y la gestión de los informes, definir políticas y proteger el acceso a los informes y a las carpetas, controlar la manera en que se procesan y distribuyen los informes y normalizar el modo en que se usan los informes en la empresa.

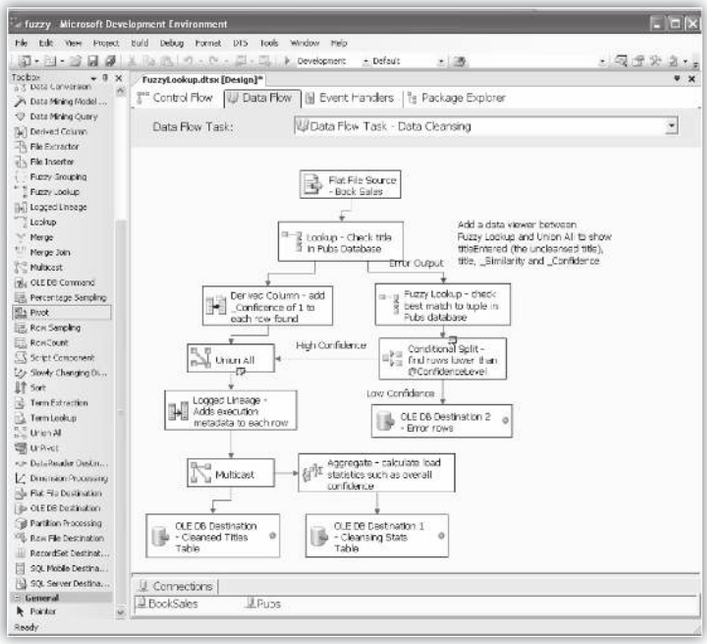


Figura 30.7. Carga de datos usando búsquedas difusas.

Notas bibliográficas

En www.microsoft.com/Downloads/Release.asp?ReleaseID=25503 se dispone de información detallada sobre el uso de los sistemas certificados C2 con SQL Server.

El entorno de optimización de SQL Server se basa en el prototipo de optimizador Cascades, que propuso Graefe [1995]. Simmen et ál. [1996] estudian el esquema para la reducción de las columnas de agrupación. Galindo-Legaria y Joshi [2001] y Elhemali et ál. [2007] presentan la variedad de estrategias de ejecución que SQL Server considera durante la optimización basada en costes de subconsultas. Chaudhuri et ál. [1999] estudian información adicional sobre los aspectos de ajuste automático de SQL Server. Chaudhuri y Shim [1994] y Yan y Larson [1995] estudian la reordenación de operaciones de agregación.

Chatziantoniou y Ross [1997] y Galindo-Legaria y Joshi [2001] propusieron la alternativa usada por SQL Server para las consultas de SQL que necesitan una autorreunión. Según este esquema, el optimizador detecta el patrón y considera la ejecución segmento por segmento. Pellenkoft et ál. [1997] estudian el esquema de optimización para la generación del espacio completo de búsqueda mediante un conjunto de transformaciones que son completas, locales y no redundantes. Graefe et ál. [1998] tratan sobre las operaciones de asociación que soportan la agregación y la reunión básicas, con varias optimizaciones, extensiones y ajustes dinámicos del sesgo de los datos. Graefe et ál. [1998] presentan la idea de reunir los índices con el único propósito de ensamblar una fila con el conjunto de columnas necesario para la consulta. Argumentan que esto a veces resulta más rápido que explorar la tabla base.

Blakeley [1996] y Blakeley y Pizzo [2001] estudian la comunicación con el motor de almacenamiento mediante OLE-DB. Blakeley et ál. [2005] detallan la implementación de las capacidades de consulta distribuida y heterogénea de SQL Server. Acheson et ál. [2004] proporcionan detalles sobre la integración de CLR de .NET dentro del proceso de SQL Server.

Blakeley et ál. [2008] describen los contratos para UDT, UDAggs y UDF con más detalle. Blakeley et ál. [2006] describen el ADO.NET Entity Framework. Melnik et ál. [2007] describen la tecnología de asociación tras el ADO.NET Entity Framework. Adya et ál. [2007] proporcionan una descripción general de la arquitectura de ADO.NET Entity Framework. La norma SQL:2003 se define en SQL/XML [2004]. Rys [2001] proporciona más detalles sobre la funcionalidad XML de SQL Server 2000. Rys [2004] ofrece una visión general de las extensiones de la agregación **for xml.** Para obtener más información sobre las capacidades XML que se pueden usar en el lado del cliente o dentro del CLR, consulte la colección de libros blancos en http://msdn.microsoft.com/XML/BuildingXML/XMLandDatabase/default.aspx. El modelo de datos XQuery 1.0/XPath 2.0 se define en Walsh et ál. [2007]. Rys [2003] proporciona una visión general de las técnicas de implementación de XQuery en el contexto de las bases de datos relacionales. El esquema de numeración OrdPath se describe en O'Neil et ál. [2004]. Pal et ál. [2004] y Baras et ál. [2005] ofrecen más información sobre el indexado de XML, la algebrización de XQuery y la optimización en SQL Server 2005.

Parte 10
Apéndices

En el Apéndice A se presentan los detalles completos de la base de datos de la universidad que se ha utilizado como ejemplo en el libro, incluyendo un diagrama E-R, el LDD de SQL y datos de ejemplo que se han utilizado en el libro. El LDD y los datos de ejemplo también están disponibles en el sitio Web del libro, db-book.com, para su uso en los ejercicios de laboratorio.

El resto de los apéndices no forman parte del libro impreso, pero están disponibles en línea en el sitio Web del libro, db-book.com (en inglés). Incluyen:

- Apéndice B (Diseño avanzado de bases de datos relacionales). Empieza con la teoría de las dependencias multivaloradas; recuerde que las dependencias multivaloradas se presentaron en el Capítulo 8. A continuación, se presenta la forma normal proyecto-reunión, que se basa en un tipo de restricciones llamadas dependencias de reunión; las dependencias de reunión son una generalización de las dependencias multivaloradas. El apéndice concluye con otra forma normal llamada forma normal de clave-dominio.
- Apéndice C (Otros lenguajes relacionales de consulta), comienza presentando el lenguaje de consulta relacional Query-by-Example (QBE), que se diseñó para su uso por no programadores. En QBE, las consultas se parecen a una colección de tablas que contienen un ejemplo de los datos que se desean obtener. A continuación se presenta el lenguaje gráfico de Microsoft Access, que está basado en QBE, seguido del lenguaje Datalog, que tiene una sintaxis similar a la del lenguaje de programación lógica Prolog.
- Apéndices D (Modelo de red) y E (Modelo jerárquico). Tratan los modelos de red y jerárquico. Estos dos modelos de datos son anteriores al modelo relacional y proporcionan un menor nivel de abstracción. Abstraen algunos, pero no todos, los detalles de las estructuras de datos reales que se utilizan para almacenar los datos en disco. Estos modelos solo se usan en algunas aplicaciones.

Para los Apéndices B-E, se muestran los conceptos usando un banco con el esquema que se muestra en la Figura 2.15.

Apéndice A

Esquema detallado de la universidad

En este apéndice se presenta el detalle completo del ejemplo de la base de datos de la universidad que se ha utilizado en el libro. En la Sección A.1 se presenta el esquema completo utilizado en el texto y el diagrama E-R que se corresponde con dicho esquema. En la Sección A.2 se presenta una definición de datos SQL relativamente completa del ejemplo. Tras la lista de tipos de datos de cada uno de los atributos se incluye un número importante de restricciones. Finalmente, en la Sección A.3 se presentan datos de ejemplo que se corresponden con el esquema. En el sitio web del libro, db-book.com, dispone de guiones de SQL para crear todas las relaciones del esquema y probarlas con los datos del ejemplo.

A.1. Esquema completo

El esquema completo de la base de datos de la universidad como se utiliza en el texto es el que se muestra en la Figura A.1. El diagrama E-R que se corresponde con este esquema, y que se usa en el texto, se muestra en la Figura A.2.

aula (*edificio*, *número_aula*, *capacidad*)

departamento (*nombre_dept*, *edificio*, *presupuesto*)

asignatura (*asignatura_id*, *nombre_asig*, *nombre_dept*, *créditos*)

profesor (*ID*, *nombre*, *nombre_dept*, *sueldo*)

sección (*asignatura_id*, *secc_id*, *semestre*, *año*, *edificio*, *número_aula*, *franja_horaria_id*)

enseña (*ID*, *asignatura_id*, *secc_id*, *semestre*, *año*)

estudiante (*ID*, *nombre*, *nombre_dept*, *tot_cred*)

matricula (*ID*, *asignatura_id*, *secc_id*, *semestre*, *año*, *nota*)

tutor (*e_ID*, *p_ID*)

franja_horaria (*franja_horaria_id*, *día*, *hora_inicio*, *hora_fin*)

prerreq (*asignatura_id*, *prerreq_id*)

Figura A.1. Esquema de base de datos de la universidad.

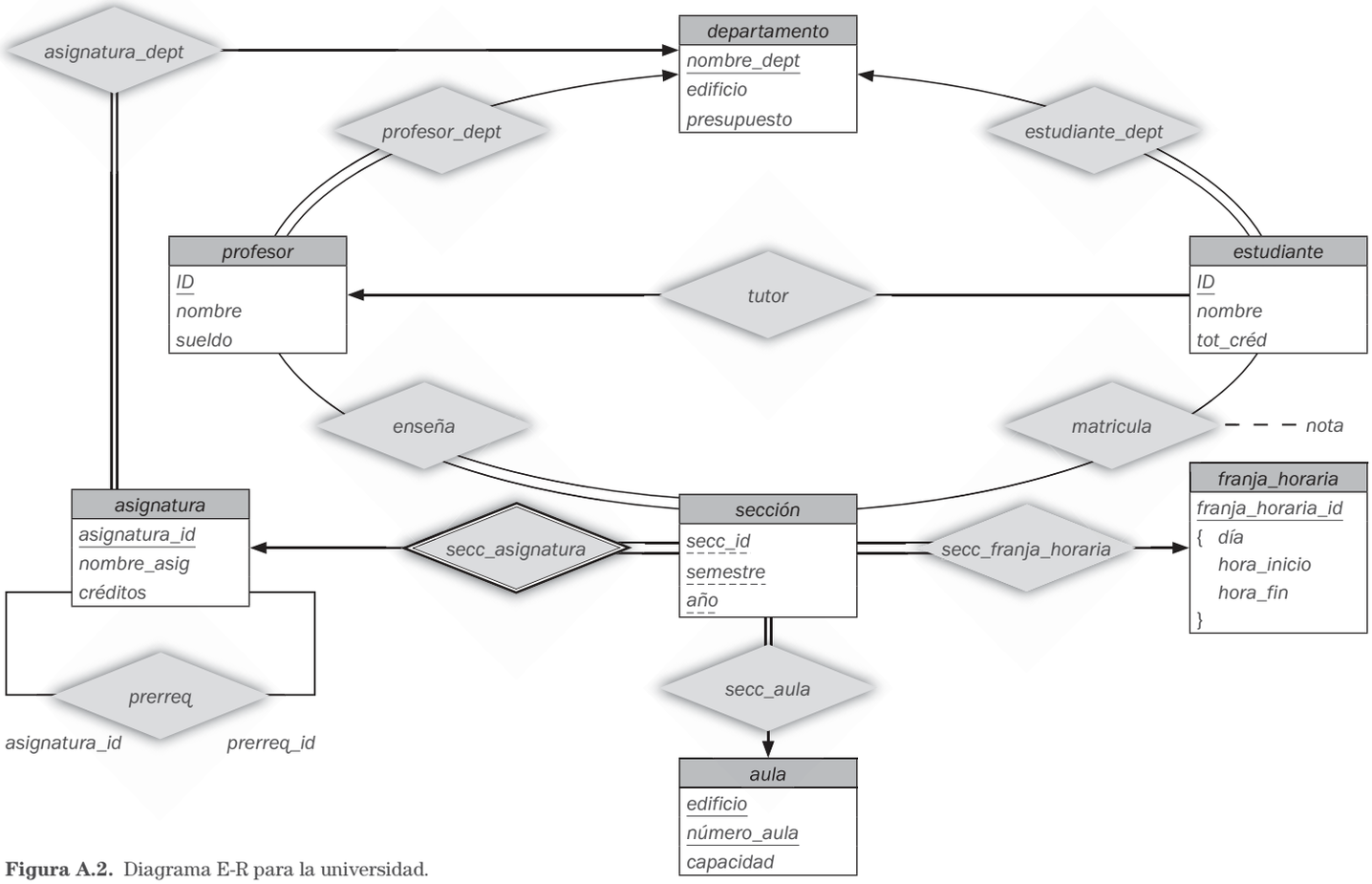


Figura A.2. Diagrama E-R para la universidad.

A.2. LDD

En esta sección se presenta una definición de datos de SQL relativamente completa del ejemplo. Tras la lista de los tipos de datos de los atributos, se incluye un número importante de restricciones.

create table *aula*
(*edificio* **varchar** (15),
número_aula **varchar** (7),
capacidad **numeric** (4,0),
primary key (*edificio*, *número_aula*));

create table *departamento*
(*nombre_dept* **varchar** (20),
edificio **varchar** (15),
presupuesto **numeric** (12,2) **check** (*presupuesto* > 0),
primary key (*nombre_dept*));

create table *asignatura*
(*asignatura_id* **varchar** (8),
nombre_asig **varchar** (50),
nombre_dept **varchar** (20),
créditos **numeric** (2,0) **check** (*créditos* > 0),
primary key (*asignatura_id*),
foreign key (*nombre_dept*) **references** *departamento* **on delete set null**);

create table *profesor*
(*ID* **varchar** (5),
nombre **varchar** (20) **not null**,
nombre_dept **varchar** (20),
sueldo **numeric** (8,2) **check** (*sueldo* > 29000),
primary key (*ID*),
foreign key (*nombre_dept*) **references** *departamento* **on delete set null**);

create table *sección*
(*asignatura_id* **varchar** (8),
secc_id **varchar** (8),
semestre **varchar** (6) **check** (*semestre* **in** ('Otoño', 'Invierno', 'Primavera', 'Verano')),
año **numeric** (4,0) **check** (*año* > 1701 **and** *año* < 2100),
edificio **varchar** (15),
número_aula **varchar** (7),
franja_horaria_id **varchar** (4),
primary key (*asignatura_id*, *secc_id*, *semestre*, *año*),
foreign key (*asignatura_id*) **references** *asignatura* **on delete cascade**,
foreign key (*edificio*, *número_aula*) **references** *aula* **on delete set null**);

En el LDD anterior se ha añadido la especificación **on delete cascade** a una restricción de clave externa si la existencia de la tupla depende de la tupla referenciada. Por ejemplo, se añade la especificación **on delete cascade** a la restricción de clave externa desde *sección* (que se genera desde la entidad débil *sección*), a *asignatura* (que es su relación identificadora). En otras restricciones de clave externa se especifica **on delete set null,** que permite el borrado de una tupla referenciada al poner el valor referenciado a *null*, o no añadir ninguna especificación, lo que evita el borrado de cualquier tupla referenciada. Por ejemplo, si se borra un departamento, no querríamos borrar a los profesores asociados; la restricción de clave externa de *profesor* a *departamento* lo que hace es establecer el atributo *nombre_dept* a null. Por otra parte, la restricción de clave externa de la relación *prerreq*, que se muestra a continuación, evita el borrado de una asignatura que se necesita como prerrequisito para otra asignatura. Para la relación *tutor*, que se muestra a continuación, se permite establecer *p_ID* a null si borra un profesor, pero se borra una tupla tutor si se borra la referencia al estudiante.

create table *enseña*
(*ID* **varchar** (5),
asignatura_id **varchar** (8),
secc_id **varchar** (8),
semestre **varchar** (6),
año **numeric** (4,0),
primary key (*ID*, *asignatura_id*, *secc_id*, *semestre*, *año*),
foreign key (*asignatura_id*, *secc_id*, *semestre*, *año*) **references** *sección* **on delete cascade**,
foreign key (*ID*) **references** *profesor* **on delete cascade**);

create table *estudiante*
(*ID* **varchar** (5),
nombre **varchar** (20) **not null**,
nombre_dept **varchar** (20),
tot_cred **numeric** (3,0) **check** (*tot_cred* >= 0),
primary key (*ID*),
foreign key (*nombre_dept*) **references** *departamento* **on delete set null**);

create table *matricula*
(*ID* **varchar** (5),
asignatura_id **varchar** (8),
secc_id **varchar** (8),
semestre **varchar** (6),
año **numeric** (4,0),
nota **varchar** (2),
primary key (*ID*, *asignatura_id*, *secc_id*, *semestre*, *año*),
foreign key (*asignatura_id*, *secc_id*, *semestre*, *año*) **references** *sección* **on delete cascade**,
foreign key (*ID*) **references** *estudiante* **on delete cascade**);

create table *tutor*
(*e_ID* **varchar** (5),
p_ID **varchar** (5),
primary key (*e_ID*),
foreign key (*p_ID*) **references** *profesor* (*ID*) **on delete set null**,
foreign key (*e_ID*) **references** *estudiante* (*ID*) **on delete cascade**);

create table *prerreq*
(*asignatura_id* **varchar**(8),
prerreq_id **varchar**(8),
primary key (*asignatura_id*, *prerreq_id*),
foreign key (*asignatura_id*) **references** *asignatura* **on delete cascade**,
foreign key (*prerreq_id*) **references** *asignatura*);

La siguiente sentencia **create table** para la tabla *franja_horaria* se puede ejecutar en la mayoría de los sistemas de bases de datos, pero no funciona en Oracle (al menos hasta la versión 11), ya que Oracle no soporta el tipo estándar de SQL **time**.

```
create table franjahoraria
  (franja_horaria_id varchar (4),
  día                varchar (1) check (día in ('L', 'M', 'X', 'J',
                                            'V', 'S', 'D')),
  hora_inicio        time,
  hora_fin           time,
  primary key        (franja_horaria_id, día, hora_inicio));
```

La sintaxis para especificar la hora en SQL se muestra en estos ejemplos: '08:30', '13:55' y '5:30 PM'. Como Oracle no admite el tipo **time,** en Oracle en su lugar se utiliza el siguiente esquema:

```
create table franjahoraria
  (franja_horaria_id varchar (4),
  día                varchar (1),
  hora_inicio        numeric (2) check (hora_inicio >= 0
                                   and hora_fin < 24),
  min_inicio         numeric (2) check (min_inicio >= 0
                                   and min_inicio < 60),
  hora_fin           numeric (2) check (hora_fin >= 0
                                   and hora_fin < 24),
  min_fin            numeric (2) check (min_fin >= 0
                                   and min_fin < 60),
  primary key        (franja_horaria_id, día, hora_inicio,
                       min_inicio));
```

La diferencia es que *hora_inicio* se ha sustituido por dos atributos *hora_inicio* y *min_inicio*, y de forma similar *hora_fin* se ha sustituido por los atributos *hora_fin* y *min_fin*. Estos atributos también tienen restricciones que aseguran que solo se representan en ellos valores de tiempo válidos. Esta versión del esquema para la *franja_horaria* funciona correctamente en todas las bases de datos, incluyendo Oracle. Tenga en cuenta que aunque Oracle soporta el tipo de datos **datetime, datetime** incluye un día, un mes y un año concretos, así como una hora, y esto no es apropiado, ya que solo se desea incluir la hora. Existen dos alternativas para dividir los atributos de hora en los componentes de hora y minutos, pero ninguna es deseable. La primera es utilizar un tipo **varchar,** pero hace que sea complicado forzar las restricciones sobre los valores de una cadena de caracteres, así como realizar comparaciones sobre la hora. La segunda alternativa es codificar la hora como un entero que represente el número de minutos (o segundos) desde medianoche, pero esta alternativa requiere código adicional para cada consulta que desease convertir estos valores entre la representación habitual de hora y la codificación como un entero. En este caso se ha elegido dividirlo en dos partes.

A.3. Datos de muestra

En esta sección se proporcionan datos de muestra para cada una de las relaciones que se han definido en las secciones previas.

edificio	*número_aula*	*capacidad*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Figura A.3. La relación *aula*.

nombre_dept	*edificio*	*presupuesto*
Biología	Watson	90000
Informática	Taylor	100000
Electrónica	Taylor	85000
Finanzas	Painter	120000
Historia	Painter	50000
Música	Packard	80000
Física	Watson	70000

Figura A.4. La relación *departamento*.

asignatura_id	*nombre*	*nombre_dept*	*créditos*
BIO-101	Introducción a la Biología	Biología	4
BIO-301	Genética	Biología	4
BIO-399	Biología computacional	Biología	3
CS-101	Introducción a la Informática	Informática	4
CS-190	Diseño de juegos	Informática	4
CS-315	Robótica	Informática	3
CS-319	Procesado de imágenes	Informática	3
CS-347	Fundamentos de bases de datos	Informática	3
EE-181	Intro. a los sistemas digitales	Electrónica	3
FIN-201	Banca de inversión	Finanzas	3
HIS-351	Historia mundial	Historia	3
MU-199	Producción de música y vídeo	Música	3
PHY-101	Fundamentos de Física	Física	4

Figura A.5. La relación *asignatura*.

ID	*nombre*	*nombre_dept*	*sueldo*
10101	Srinivasan	Informática	65000
12121	Wu	Finanzas	90000
15151	Mozart	Música	40000
22222	Einstein	Física	95000
32343	El Said	Historia	60000
33456	Gold	Física	87000
45565	Katz	Informática	75000
58583	Califieri	Historia	62000
76543	Singh	Finanzas	80000
76766	Crick	Biología	72000
83821	Brandt	Informática	92000
98345	Kim	Electrónica	80000

Figura A.6. La relación *profesor*.

asignatura_id	*secc_id*	*semestre*	*año*	*edificio*	*aula*	*franja_horaria*
BIO-101	1	Verano	2009	Painter	514	B
BIO-301	1	Verano	2010	Painter	514	A
CS-101	1	Otoño	2009	Packard	101	H
CS-101	1	Primavera	2010	Packard	101	F
CS-190	1	Primavera	2009	Taylor	3128	E
CS-190	2	Primavera	2009	Taylor	3128	A
CS-315	1	Primavera	2010	Watson	120	D
CS-319	1	Primavera	2010	Watson	100	B
CS-319	2	Primavera	2010	Taylor	3128	C
CS-347	1	Otoño	2009	Taylor	3128	A
EE-181	1	Primavera	2009	Taylor	3128	C
FIN-201	1	Primavera	2010	Packard	101	B
HIS-351	1	Primavera	2010	Painter	514	C
MU-199	1	Primavera	2010	Packard	101	D
PHY-101	1	Otoño	2009	Watson	100	A

Figura A.7. La relación *sección*.

ID	asignatura_id	secc_id	semestre	año
10101	CS-101	1	Otoño	2009
10101	CS-315	1	Primavera	2010
10101	CS-347	1	Otoño	2009
12121	FIN-201	1	Primavera	2010
15151	MU-199	1	Primavera	2010
22222	PHY-101	1	Otoño	2009
32343	HIS-351	1	Primavera	2010
45565	CS-101	1	Primavera	2010
45565	CS-319	1	Primavera	2010
76766	BIO-101	1	Verano	2009
76766	BIO-301	1	Verano	2010
83821	CS-190	1	Primavera	2009
83821	CS-190	2	Primavera	2009
83821	CS-319	2	Primavera	2010
98345	EE-181	1	Primavera	2009

Figura A.8. La relación *enseña*.

ID	nombre	nombre_dept	tot_créditos
00128	Zhang	Informática	102
12345	Shankar	Informática	32
19991	Brandt	Historia	80
23121	Chávez	Finanzas	110
44553	Peltier	Física	56
45678	Levy	Física	46
54321	Williams	Informática	54
55739	Sánchez	Música	38
70557	Snow	Física	0
76543	Brown	Informática	58
76653	Aoi	Electrónica	60
98765	Bourikas	Electrónica	98
98988	Tanaka	Biología	120

Figura A.9. La relación *estudiante*.

ID	asignatura_id	secc_id	semestre	año	nota
00128	CS-101	1	Otoño	2009	A
00128	CS-347	1	Otoño	2009	A−
12345	CS-101	1	Otoño	2009	C
12345	CS-190	2	Primavera	2009	A
12345	CS-315	1	Primavera	2010	A
12345	CS-347	1	Otoño	2009	A
19991	HIS-351	1	Primavera	2010	B
23121	FIN-201	1	Primavera	2010	C+
44553	PHY-101	1	Otoño	2009	B−
45678	CS-101	1	Otoño	2009	F
45678	CS-101	1	Primavera	2010	B+
45678	CS-319	1	Primavera	2010	B
54321	CS-101	1	Otoño	2009	A−
54321	CS-190	2	Primavera	2009	B+
55739	MU-199	1	Primavera	2010	A−
76543	CS-101	1	Otoño	2009	A
76543	CS-319	2	Primavera	2010	A
76653	EE-181	1	Primavera	2009	C
98765	CS-101	1	Otoño	2009	C−
98765	CS-315	1	Primavera	2010	B
98988	BIO-101	1	Verano	2009	A
98988	BIO-301	1	Verano	2010	*null*

Figura A.10. La relación *matricula*.

e_ID	*p_ID*
00128	45565
12345	10101
23121	76543
44553	22222
45678	22222
76543	45565
76653	98345
98765	98345
98988	76766

Figura A.11. La relación *tutor*.

asignatura_id	*prerreq_id*
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Figura A.12. La relación *prerreq*.

franja_horaria_id	*día*	*hora_inicio*	*hora_fin*
A	L	8:00	8:50
A	X	8:00	8:50
A	V	8:00	8:50
B	L	9:00	9:50
B	X	9:00	9:50
B	V	9:00	9:50
C	L	11:00	11:50
C	X	11:00	11:50
C	V	11:00	11:50
D	L	13:00	13:50
D	X	13:00	13:50
D	V	13:00	13:50
E	M	10:30	11:45
E	J	10:30	11:45
F	M	14:30	15:45
F	J	14:30	15:45
G	L	16:00	16:50
G	X	16:00	16:50
G	V	16:00	16:50
H	X	10:00	12:30

Figura A.13. La relación *franja_horaria*.

franja_horaria_id	*día*	*hora_inicio*	*min_inicio*	*hora_fin*	*min_fin*
A	L	8	0	8	50
A	X	8	0	8	50
A	V	8	0	8	50
B	L	9	0	9	50
B	X	9	0	9	50
B	V	9	0	9	50
C	L	11	0	11	50
C	X	11	0	11	50
C	V	11	0	11	50
D	L	13	0	13	50
D	X	13	0	13	50
D	V	13	0	13	50
E	M	10	30	11	45
E	J	10	30	11	45
F	M	14	30	15	45
F	J	14	30	15	45
G	L	16	0	16	50
G	X	16	0	16	50
G	V	16	0	16	50
H	X	10	0	12	30

Figura A.14. La relación *franja_horaria* con la hora de inicio y fin separadas en horas y minutos.

Bibliografía

[Abadi 2009] D. Abadi, «Data Management in the Cloud: Limitations and Opportunities», *Data Engineering Bulletin*, Volumen 32, Número 1 (2009), páginas 3–12.

[Abadi et ál. 2008] D. J. Abadi, S. Madden y N. Hachem, «Column-stores vs. row-stores: how different are they really?», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2008), páginas 967–980.

[Abiteboul et ál. 1995] S. Abiteboul, R. Hull y V. Vianu, *Foundations of Databases*, Addison Wesley (1995).

[Abiteboul et ál. 2003] S. Abiteboul, R. Agrawal, P. A. Bernstein, M. J. Carey, et ál. «The Lowell Database Research Self Assessment» (2003).

[Acheson et ál. 2004] A. Acheson, M. Bendixen, J. A. Blakeley, I. P. Carlin, E. Ersan, J. Fang, X. Jiang, C. Kleinerman, B. Rathakrishnan, G. Schaller, B. Sezgin, R. Venkatesh y H. Zhang, «Hosting the .NET Runtime in Microsoft SQL Server», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2004), páginas 860–865.

[Adali et ál. 1996] S. Adali, K. S. Candan, Y. Papakonstantinou y V. S. Subrahmanian, «Query Caching and Optimization in Distributed Mediator Systems», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 137–148.

[Adya et ál. 2007] A. Adya, J. A. Blakeley, S. Melnik y S. Muralidhar, «Anatomy of the ADO.NET entity framework», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2007), páginas 877–888.

[Agarwal et ál. 1996] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan y S. Sarawagi, «On the Computation of Multidimensional Attributes», *Proc. of the International Conf. on Very Large Databases* (1996), páginas 506–521.

[Agrawal and Srikant 1994] R. Agrawal and R. Srikant, «Fast Algorithms for Mining Association Rules in Large Databases», *Proc. of the International Conf. on Very Large Databases* (1994), páginas 487–499.

[Agrawal et ál. 1992] R. Agrawal, S. P. Ghosh, T. Imielinski, B. R. Iyer y A. N. Swami, «An Interval Classifier for Database Mining Applications», *Proc. of the International Conf. on Very Large Databases* (1992), páginas 560–573.

[Agrawal et ál. 1993a] R. Agrawal, T. Imielinski y A. Swami, «Mining Association Rules between Sets of Items in Large Databases», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993).

[Agrawal et ál. 1993b] R. Agrawal, T. Imielinski y A. N. Swami, «Database Mining: A Performance Perspective», *IEEE Transactions on Knowledge and Data Engineering*, Volumen 5, Número 6 (1993), páginas 914–925.

[Agrawal et ál. 2000] S. Agrawal, S. Chaudhuri y V. R. Narasayya, «Automated Selection of Materialized Views and Indexes in SQL Databases», *Proc. of the International Conf. on Very Large Databases* (2000), páginas 496–505.

[Agrawal et ál. 2002] S. Agrawal, S. Chaudhuri y G. Das, «DBXplorer: A System for Keyword-Based Search over Relational Databases», *Proc. of the International Conf. on Data Engineering* (2002).

[Agrawal et ál. 2004] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya y M. Syamala, «Database Tuning Advisor for Microsoft SQL Server 2005», *Proc. of the International Conf. on Very Large Databases* (2004).

[Agrawal et ál. 2009] R. Agrawal, A. Ailamaki, P. A. Bernstein, E. A. Brewer, M. J. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. J. Franklin, H. García-Molina, J. Gehrke, L. Gruenwald, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. F. Korth, D. Kossmann, S. Madden, R. Magoulas, B. C. Ooi, T. O^Reilly, R. Ramakrishnan, S. Sarawagi y G. W. Michael Stonebraker, Alexander S. Szalay, «The Claremont Report on Database Research», Communications of the ACM, Volumen 52, Número 6 (2009), páginas 56–65.

[Ahmed et ál. 2006] R. Ahmed, A. Lee, A. Witkowski, D. Das, H. Su, M. Za¨ıt y T. Cruanes, «Cost-Based Query Transformation in Oracle», *Proc. of the International Conf. on Very Large Databases* (2006), páginas 1026–1036.

[Aho et ál. 1979a] A. V. Aho, C. Beeri y J. D. Ullman, «The Theory of Joins in Relational Databases», *ACM Transactions on Database Systems*, Volumen 4, Número 3 (1979), páginas 297–314.

[Aho et ál. 1979b] A. V. Aho, Y. Sagiv y J. D. Ullman, «Equivalences among Relational Expressions», *SIAM Journal of Computing*, Volumen 8, Número 2 (1979), páginas 218–246.

[Ailamaki et ál. 2001] A. Ailamaki, D. J. DeWitt, M. D. Hill y M. Skounakis, «Weaving Relations for Cache Performance», *Proc. of the International Conf. on Very Large Databases* (2001), páginas 169–180.

[Alonso and Korth 1993] R. Alonso and H. F. Korth, «Database System Issues in Nomadic Computing», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993), páginas 388–392.

[Amer-Yahia et ál. 2004] S. Amer-Yahia, C. Botev y J. Shanmugasundaram, «TeXQuery: A Full-Text Search Extension to XQuery», *Proc. of the International World Wide Web Conf.* (2004).

[Anderson et ál. 1992] D. P. Anderson, Y. Osawa y R. Govindan, «A File System for Continuous Media», *ACM Transactions on Database Systems*, Volumen 10, Number 4 (1992), páginas 311–337.

[Anderson et ál. 1998] T. Anderson, Y. Breitbart, H. F. Korth y A. Wool, «Replication, Consistency and Practicality: Are These Mutually Exclusive?», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1998).

[ANSI 1986] *American National Standard for Information Systems: Database Language SQL*. American National Standards Institute (1986).

[ANSI 1989] *Database Language SQL with Integrity Enhancement, ANSI X3, 135–1989*. American National Standards Institute, New York (1989).

[ANSI 1992] *Database Language SQL, ANSI X3,135–1992*. American National Standards Institute, New York (1992).

[Antoshenkov 1995] G. Antoshenkov, «Byte-aligned Bitmap Compression (poster abstract)», *IEEE Data Compression Conf.* (1995).

[Appelt and Israel 1999] D. E. Appelt and D. J. Israel, «Introduction to Information Extraction Technology», *Proc. of the International Joint Conferences on Artificial Intelligence* (1999).

[Apt and Pugin 1987] K. R. Apt and J. M. Pugin, «Maintenance of Stratified Database Viewed as a Belief Revision System», *Proc. of the ACM Symposium on Principles of Database Systems* (1987), páginas 136–145.

[Armstrong 1974] W. W. Armstrong, «Dependency Structures of Data Base Relationships», *Proc. of the 1974 IFIP Congress* (1974), páginas 580–583.

[Astrahan et ál. 1976] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade y V. Watson, «System R, A Relational Approach to Data Base Management», *ACM Transactions on Database Systems*, Volume 1, Número 2 (1976), páginas 97–137.

[Atreya et ál. 2002] M. Atreya, B. Hammond, S. Paine, P. Starrett y S. Wu, *Digital Signatures*, RSA Press (2002).

[Atzeni and Antonellis 1993] P. Atzeni and V. D. Antonellis, *Relational Database Theory*, Benjamin Cummings (1993).

[Baeza-Yates and Ribeiro-Neto 1999] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, Addison Wesley (1999).

[Bancilhon et ál. 1989] F. Bancilhon, S. Cluet y C. Delobel, «A Query Language for the O_2 Object-Oriented Database», *Proc. of the Second Workshop on Database Programming Languages* (1989).

[Baras et ál. 2005] A. Baras, D. Churin, I. Cseri, T. Grabs, E. Kogan, S. Pal, M. Rys y O. Seeliger. «Implementing XQuery in a Relational Database System» (2005).

[Baru et ál. 1995] C. Baru et ál., «DB2 Parallel Edition», *IBM Systems Journal*, Volume 34, Número 2 (1995), páginas 292–322.

[Bassiouni 1988] M. Bassiouni, «Single-site and Distributed Optimistic Protocols for Concurrency Control», *IEEE Transactions on Software Engineering*, Volumen SE-14, Número 8 (1988), páginas 1071–1080.

[Batini et ál. 1992] C. Batini, S. Ceri y S. Navathe, *Database Design: An Entity-Relationship Approach*, Benjamin Cummings (1992).

[Bayer 1972] R. Bayer, «Symmetric Binary B-trees: Data Structure and Maintenance Algorithms», *Acta Informatica*, Volumen 1, Número 4 (1972), páginas 290–306.

[Bayer and McCreight 1972] R. Bayer and E. M. McCreight, «Organization and Maintenance of Large Ordered Indices», *Acta Informatica*, Volumen 1, Número 3 (1972), páginas 173–189.

[Bayer and Schkolnick 1977] R. Bayer and M. Schkolnick, «Concurrency of Operating on B-trees», *Acta Informatica*, Volumen 9, Número 1 (1977), páginas 1–21.

[Bayer and Unterauer 1977] R. Bayer and K. Unterauer, «Prefix B-trees», *ACM Transactions on Database Systems*, Volumen 2, Número 1 (1977), páginas 11–26.

[Bayer et ál. 1978] R. Bayer, R. M. Graham y G. Seegmuller, editors, *Operating Systems: An Advanced Course*, Springer Verlag (1978).

[Beckmann et ál. 1990] N. Beckmann, H. P. Kriegel, R. Schneider y B. Seeger, «The R*-tree: An Efficient and Robust Access Method for Points and Rectangles», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990), páginas 322–331.

[Beeri et ál. 1977] C. Beeri, R. Fagin y J. H. Howard, «A Complete Axiomatization for Functional and Multivalued Dependencies», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1977), páginas 47–61.

[Bentley 1975] J. L. Bentley, «Multidimensional Binary Search Trees Used for Associative Searching», *Communications of the ACM*, Volumen 18, Número 9 (1975), páginas 509–517.

[Berenson et ál. 1995] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil y P. O'Neil, «A Critique of ANSI SQL Isolation Levels», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 1–10.

[Bernstein and Goodman 1981] P. A. Bernstein and N. Goodman, «Concurrency Control in Distributed Database Systems», *ACM Computing Survey*, Volumen 13, Número 2 (1981), páginas 185–221.

[Bernstein and Newcomer 1997] P. A. Bernstein and E. Newcomer, *Principles of Transaction Processing*, Morgan Kaufmann (1997).

[Bernstein et ál. 1998] P. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. García-Molina, J. Gray, J. Held, J. Hellerstein, H. V. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker y J. Ullman, «The Asilomar Report on Database Research», *ACM SIGMOD Record*, Volumen 27, Número 4 (1998).

[Berson et ál. 1995] S. Berson, L. Golubchik y R. R. Muntz, «Fault Tolerant Design of Multimedia Servers», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 364–375.

[Bhalotia et ál. 2002] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti y S. Sudarshan, «Keyword Searching and Browsing in Databases using BANKS», *Proc. of the International Conf. on Data Engineering* (2002).

[Bharat and Henzinger 1998] K. Bharat and M. R. Henzinger, «Improved Algorithms for Topic Distillation in a Hyperlinked Environment», *Proc. of the ACM SIGIR Conf. on Research and Development in Information Retrieval* (1998), páginas 104–111.

[Bhattacharjee et ál. 2003] B. Bhattacharjee, S. Padmanabhan, T. Malkemus, T. Lai, L. Cranston y M. Huras, «Efficient Query Processing for Multi-Dimensionally Clustered Tables in DB2», *Proc. of the International Conf. on Very Large Databases* (2003), páginas 963–974.

[Biskup et ál. 1979] J. Biskup, U. Dayal y P. A. Bernstein, «Synthesizing Independent Database Schemas», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1979), páginas 143–152.

[Bitton et ál. 1983] D. Bitton, D. J. DeWitt, and C. Turbyfill, «Benchmarking Database Systems: A Systematic Approach», *Proc. of the International Conf. on Very Large Databases* (1983).

[Blakeley 1996] J. A. Blakeley, «Data Access for the Masses through OLE DB», In *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 161–172.

[Blakeley and Pizzo 2001] J. A. Blakeley and M. Pizzo, «Enabling Component Databases with OLE DB», In K. R. Dittrich and A. Geppert, editors, *Component Database Systems*, Morgan Kaufmann Publishers (2001), páginas 139–173.

[Blakeley et ál. 1986] J. A. Blakeley, P. Larson y F. W. Tompa, «Efficiently Updating Materialized Views», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1986), páginas 61–71.

[Blakeley et ál. 2005] J. A. Blakeley, C. Cunningham, N. Ellis, B. Rathakrishnan y M.-C. Wu, «Distributed/Heterogeneous Query Processing in Microsoft SQL Server», *Proc. of the International Conf. on Data Engineering* (2005).

[Blakeley et ál. 2006] J. A. Blakeley, D. Campbell, S. Muralidhar y A. Nori, «The ADO.NET entity framework: making the conceptual level real», *SIGMOD Record*, Volumen 35, Número 4 (2006), páginas 32–39.

[Blakeley et ál. 2008] J. A. Blakeley, V. Rao, I. Kunen, A. Prout, M. Henaire, and C. Kleinerman, «.NET database programmability and extensibility in Microsoft SQL server», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2008), páginas 1087–1098.

[Blasgen and Eswaran 1976] M. W. Blasgen and K. P. Eswaran, «On the Evaluation of Queries in a Relational Database System», *IBM Systems Journal*, Volumen 16, (1976), páginas 363–377.

[Boyce et ál. 1975] R. Boyce, D. D. Chamberlin, W. F. King y M. Hammer, «Specifying Queries as Relational Expressions», *Communications of the ACM*, Volumen 18, Número 11 (1975), páginas 621–628.

[Brantner et ál. 2008] M. Brantner, D. Florescu, D. Graf, D. Kossmann, and T. Kraska, «Building a Database on S3», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2008), páginas 251–263.

[Breese et ál. 1998] J. Breese, D. Heckerman y C. Kadie, «Empirical Analysis of Predictive Algorithms for Collaborative Filtering», *Procs. Conf. on Uncertainty in Artificial Intelligence*, Morgan Kaufmann (1998).

[Breitbart et ál. 1999a] Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri y A. Silberschatz, «Update Propagation Protocols For Replicated Databases», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1999), páginas 97–108.

[Breitbart et ál. 1999b] Y. Breitbart, H. Korth, A. Silberschatz y S. Sudarshan, «Distributed Databases», *Encyclopedia of Electrical and Electronics Engineering*, John Wiley and Sons (1999).

[Brewer 2000] E. A. Brewer, «Towards robust distributed systems (abstract)», *Proc. of the ACM Symposium on Principles of Distributed Computing* (2000), página 7.

[Brin and Page 1998] S. Brin and L. Page, «The Anatomy of a Large-Scale Hypertextual Web Search Engine», *Proc. of the International World Wide Web Conf.* (1998).

[Brinkhoff et ál. 1993] T. Brinkhoff, H.-P. Kriegel y B. Seeger, «Efficient Processing of Spatial Joins Using R-trees», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993), páginas 237–246.

[Bruno et ál. 2002] N. Bruno, S. Chaudhuri y L. Gravano, «Top-k Selection Queries Over Relational Databases: Mapping Strategies and Performance Evaluation», *ACM Transactions on Database Systems*, Volumen 27, Número 2 (2002), páginas 153–187.

[Buckley and Silberschatz 1983] G. Buckley and A. Silberschatz, «Obtaining Progressive Protocols for a Simple Multiversion Database Model», *Proc. of the International Conf. on Very Large Databases* (1983), páginas 74–81.

[Buckley and Silberschatz 1984] G. Buckley and A. Silberschatz, «Concurrency Control in Graph Protocols by Using Edge Locks», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1984), páginas 45–50.

[Buckley and Silberschatz 1985] G. Buckley and A. Silberschatz, «Beyond TwoPhase Locking», *Journal of the ACM*, Volumen 32, Número 2 (1985), páginas 314–326.

[Bulmer 1979] M. G. Bulmer, *Principles of Statistics*, Dover Publications (1979).

[Burkhard 1976] W. A. Burkhard, «Hashing and Trie Algorithms for Partial Match Retrieval», *ACM Transactions on Database Systems*, Volumen 1, Número 2 (1976), páginas 175–187.

[Burkhard 1979] W. A. Burkhard, «Partial-match Hash Coding: Benefits of Redundancy», *ACM Transactions on Database Systems*, Volumen 4, Número 2 (1979), páginas 228–239.

[Cannan and Otten 1993] S. Cannan and G. Otten, *SQL — The Standard Handbook*, McGra-Hill (1993).

[Carey 1983] M. J. Carey, «Granularity Hierarchies in Concurrency Control», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1983), páginas 156–165.

[Carey and Kossmann 1998] M. J. Carey and D. Kossmann, «Reducing the Braking Distance of an SQL Query Engine», *Proc. of the International Conf. on Very Large Databases* (1998), páginas 158–169.

[Carey et ál. 1991] M. Carey, M. Franklin, M. Livny y E. Shekita, «Data Caching Tradeoffs in Client-Server DBMS Architectures», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1991).

[Carey et ál. 1993] M. J. Carey, D. DeWitt y J. Naughton, «The OO7 Benchmark», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993).

[Carey et ál. 1999] M. J. Carey, D. D. Chamberlin, S. Narayanan, B. Vance, D. Doole, S. Rielau, R. Swagerman y N. Mattos, «O-O, What Have They Done to DB2?», *Proc. of the International Conf. on Very Large Databases* (1999), páginas 542–553.

[Cattell 2000] R. Cattell, editor, *The Object Database Standard: ODMG 3.0*, Morgan Kaufmann (2000).

[Cattell and Skeen 1992] R. Cattell and J. Skeen, «Object Operations Benchmark», *ACM Transactions on Database Systems*, Volumen 17, Número 1 (1992).

[Chakrabarti 1999] S. Chakrabarti, «Recent Results in Automatic Web Resource Discovery», *ACM Computing Surveys*, Volumen 31, Número 4 (1999).

[Chakrabarti 2000] S. Chakrabarti, «Data Mining for Hypertext: A Tutorial Survey», *SIGKDD Explorations*, Volumen 1, Número 2 (2000), páginas 1–11.

[Chakrabarti 2002] S. Chakrabarti, *Mining the Web: Discovering Knowledge from HyperText Data*, Morgan Kaufmann (2002).

[Chakrabarti et ál. 1998] S. Chakrabarti, S. Sarawagi y B. Dom, «Mining Surprising Patterns Using Temporal Description Length», *Proc. of the International Conf. on Very Large Databases* (1998), páginas 606–617.

[Chakrabarti et ál. 1999] S. Chakrabarti, M. van den Berg y B. Dom, «Focused Crawling: A New Approach to Topic Specific Web Resource Discovery», *Proc. of the International World Wide Web Conf.* (1999).

[Chamberlin 1996] D. Chamberlin, *Using the New DB2: IBM's Object-Relational Database System*, Morgan Kaufmann (1996).

[Chamberlin 1998] D. D. Chamberlin, *A Complete Guide to DB2 Universal Database*, Morgan Kaufmann (1998).

[Chamberlin and Boyce 1974] D. D. Chamberlin and R. F. Boyce, «SEQUEL: A Structured English Query Language», *ACM SIGMOD Workshop on Data Description, Access, and Control* (1974), páginas 249–264.

[Chamberlin et ál. 1976] D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner y B. W. Wade, «SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control», *IBM Journal of Research and Development*, Volumen 20, Número 6 (1976), páginas 560–575.

[Chamberlin et ál. 1981] D. D. Chamberlin, M. M. Astrahan, M. W. Blasgen, J. N. Gray, W. F. King, B. G. Lindsay, R. A. Lorie, J. W. Mehl, T. G. Price, P. G. Selinger, M. Schkolnick, D. R. Slutz, I. L. Traiger, B. W. Wade y R. A. Yost, «A History and Evaluation of System R», *Communications of the ACM*, Volumen 24, Número 10 (1981), páginas 632–646.

[Chamberlin et ál. 2000] D. D. Chamberlin, J. Robie y D. Florescu, «Quilt: An XML Query Language for Heterogeneous Data Sources», *Proc. of the International Workshop on the Web and Databases (WebDB)* (2000), páginas 53–62.

[Chan and Ioannidis 1998] C.-Y. Chan and Y. E. Ioannidis, «Bitmap Index Design and Evaluation», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1998).

[Chan and Ioannidis 1999] C.-Y. Chan and Y. E. Ioannidis, «An Efficient Bitmap Encoding Scheme for Selection Queries», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1999).

[Chandra and Harel 1982] A. K. Chandra and D. Harel, «Structure and Complexity of Relational Queries», *Journal of Computer and System Sciences*, Volumen 15, Número 10 (1982), páginas 99–128.

[Chandrasekaran et ál. 2003] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss y M. Shah, «TelegraphCQ: Continuous Dataflow Processing for an Uncertain World», *First Biennial Conference on Innovative Data Systems Research* (2003).

[Chang et ál. 2008] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes y R. E. Gruber, «Bigtable: A Distributed Storage System for Structured Data», *ACM Trans. Comput. Syst.*, Volumen 26, Número 2 (2008).

[Chatziantoniou and Ross 1997] D. Chatziantoniou and K. A. Ross, «Groupwise Processing of Relational Queries», *Proc. of the International Conf. on Very Large Databases* (1997), páginas 476–485.

[Chaudhuri and Narasayya 1997] S. Chaudhuri and V. Narasayya, «An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server», *Proc. of the International Conf. on Very Large Databases* (1997).

[Chaudhuri and Shim 1994] S. Chaudhuri and K. Shim, «Including Group-By in Query Optimization», *Proc. of the International Conf. on Very Large Databases* (1994).

[Chaudhuri et ál. 1995] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim, «Optimizing Queries with Materialized Views», *Proc. of the International Conf. on Data Engineering* (1995).

[Chaudhuri et ál. 1998] S. Chaudhuri, R. Motwani y V. Narasayya, «Random sampling for histogram construction: how much is enough?», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1998), páginas 436–447.

[Chaudhuri et ál. 1999] S. Chaudhuri, E. Christensen, G. Graefe, V. Narasayya y M. Zwilling, «Self Tuning Technology in Microsoft SQL Server», *IEEE Data Engineering Bulletin*, Volumen 22, Número 2 (1999).

[Chaudhuri et ál. 2003] S. Chaudhuri, K. Ganjam, V. Ganti y R. Motwani, «Robust and Efficient Fuzzy Match for Online Data Cleaning», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2003).

[Chen 1976] P. P. Chen, «The Entity-Relationship Model: Toward a Unified View of Data», *ACM Transactions on Database Systems*, Volumen 1, Número 1 (1976), páginas 9–36.

[Chen et ál. 1994] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz y D. A. Patterson, «RAID: High-Performance, Reliable Secondary Storage», *ACM Computing Survey*, Volumen 26, Número 2 (1994).

[Chen et ál. 2007] S. Chen, A. Ailamaki, P. B. Gibbons y T. C. Mowry, «Improving hash join performance through prefetching», *ACM Transactions on Database Systems*, Volumen 32, Número 3 (2007).

[Chomicki 1995] J. Chomicki, «Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding», *ACM Transactions on Database Systems*, Volumen 20, Número 2 (1995), páginas 149–186.

[Chou and Dewitt 1985] H. T. Chou and D. J. Dewitt, «An Evaluation of Buffer Management Strategies for Relational Database Systems», *Proc. of the International Conf. on Very Large Databases* (1985), páginas 127–141.

[Cieslewicz et ál. 2009] J. Cieslewicz, W. Mee y K. A. Ross, «Cache-Conscious Buffering for Database Operators with State», *Proc. Fifth International Workshop on Data Management on New Hardware (DaMoN 2009)* (2009).

[Cochrane et ál. 1996] R. Cochrane, H. Pirahesh y N. M. Mattos, «Integrating Triggers and Declarative Constraints in SQL Database Sytems», *Proc. of the International Conf. on Very Large Databases* (1996), páginas 567–578.

[Codd 1970] E. F. Codd, «A Relational Model for Large Shared Data Banks», *Communications of the ACM*, Volumen 13, Número 6 (1970), páginas 377–387.

[Codd 1972] E. F. Codd. «Further Normalization of the Data Base Relational Model», *Rustin [1972]*, páginas 33–64 (1972).

[Codd 1979] E. F. Codd, «Extending the Database Relational Model to Capture More Meaning», *ACM Transactions on Database Systems*, Volumen 4, Número 4 (1979), páginas 397–434.

[Codd 1982] E. F. Codd, «The 1981 ACM Turing Award Lecture: Relational Database: A Practical Foundation for Productivity», *Communications of the ACM*, Volumen 25, Número 2 (1982), páginas 109–117.

[Codd 1990] E. F. Codd, *The Relational Model for Database Management: Version 2*, Addison Wesley (1990).

[Comer 1979] D. Comer, «The Ubiquitous B-tree», *ACM Computing Survey*, Volume 11, Número 2 (1979), páginas 121–137.

[Comer 2009] D. E. Comer, *Computer Networks and Internets*, 5th edition, Prentice Hall (2009).

[Cook 1996] M. A. Cook, *Building Enterprise Information Architecture: Reengineering Information Systems*, Prentice Hall (1996).

[Cooper et ál. 2008] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver y R. Yerneni, «PNUTS: Yahoo!'s hosted data serving platform», *Proceedings of the VLDB Endowment*, Volumen 1, Número 2 (2008), páginas 1277–1288.

[Cormen et ál. 1990] T. Cormen, C. Leiserson y R. Rivest, *Introduction to Algorithms*, MIT Press (1990).

[Cortes and Vapnik 1995] C. Cortes and V. Vapnik, *Machine Learning*, Volumen 20, Número 3 (1995), páginas 273–297.

[Cristianini and Shawe-Taylor 2000] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and other Kernel-Based Learning Methods*, Cambridge University Press (2000).

[Dageville and Za¨ıt 2002] B. Dageville and M. Za¨ıt, «SQL Memory Management in Oracle9i», *Proc. of the International Conf. on Very Large Databases* (2002), páginas 962–973.

[Dageville et ál. 2004] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Za¨ıt y M. Ziauddin, «Automatic SQL Tuning in Oracle 10g», *Proc. of the International Conf. on Very Large Databases* (2004), páginas 1098–1109.

[Dalvi et ál. 2009] N. Dalvi, R. Kumar, B. Pang, R. Ramakrishnan, A. Tomkins, P. Bohannon, S. Keerthi y S. Merugu, «A Web of Concepts», *Proc. of the ACM Symposium on Principles of Database Systems* (2009).

[Daniels et ál. 1982] D. Daniels, P. G. Selinger, L. M. Haas, B. G. Lindsay, C. Mohan, A. Walker y P. F. Wilms. «An Introduction to Distributed Query Compilation in R*», *Schneider [1982]* (1982).

[Dashti et ál. 2003] A. Dashti, S. H. Kim, C. Shahabi y R. Zimmermann, *Streaming Media Server Design*, Prentice Hall (2003).

[Date 1983] C. J. Date, «The Outer Join», *Proc. of the International Conference on Databases*, John Wiley and Sons (1983), páginas 76–106.

[Date 1989] C. Date, *A Guide to DB2*, Addison Wesley (1989).

[Date 1993] C. J. Date, «How SQL Missed the Boat», *Database Programming and Design*, Volumen 6, Número 9 (1993).

[Date 2003] C. J. Date, *An Introduction to Database Systems*, 8th edition, Addison Wesley (2003).

[Date and Darwen 1997] C. J. Date and G. Darwen, *A Guide to the SQL Standard*, 4th edition, Addison Wesley (1997).

[Davis et ál. 1983] C. Davis, S. Jajodia, P. A. Ng, and R. Yeh, editors, *EntityRelationship Approach to Software Engineering*, North Holland (1983).

[Davison and Graefe 1994] D. L. Davison and G. Graefe, «Memory-Contention Responsive Hash Joins», *Proc. of the International Conf. on Very Large Databases* (1994).

[Dayal 1987] U. Dayal, «Of Nests and Trees: A Unified Approach to Processing Queries that Contain Nested Subqueries, Aggregates and Quantifiers», *Proc. of the International Conf. on Very Large Databases* (1987), páginas 197–208.

[Deutsch et ál. 1999] A. Deutsch, M. Fernández, D. Florescu, A. Levy y D. Suciu, «A Query Language for XML», *Proc. of the International World Wide Web Conf.* (1999).

[DeWitt 1990] D. DeWitt, «The Gamma Database Machine Project», *IEEE Transactions on Knowledge and Data Engineering*, Volumen 2, Número 1 (1990).

[DeWitt and Gray 1992] D. DeWitt and J. Gray, «Parallel Database Systems: The Future of High Performance Database Systems», *Communications of the ACM*, Volumen 35, Número 6 (1992), páginas 85–98.

[DeWitt et ál. 1992] D. DeWitt, J. Naughton, D. Schneider y S. Seshadri, «Practical Skew Handling in Parallel Joins», *Proc. of the International Conf. on Very Large Databases* (1992).

[Dias et ál. 1989] D. Dias, B. Iyer, J. Robinson y P. Yu, «Integrated Concurrency-Coherency Controls for Multisystem Data Sharing», *Software Engineering*, Volumen 15, Número 4 (1989), páginas 437–448.

[Donahoo and Speegle 2005] M. J. Donahoo and G. D. Speegle, *SQL: Practical Guide for Developers*, Morgan Kaufmann (2005).

[Douglas and Douglas 2003] K. Douglas and S. Douglas, *PostgreSQL*, Sam's Publishing (2003).

[Dubois and Thakkar 1992] M. Dubois and S. Thakkar, editors, *Scalable Shared Memory Multiprocessors*, Kluwer Academic Publishers (1992).

[Duncan 1990] R. Duncan, «A Survey of Parallel Computer Architectures», *IEEE Computer*, Volumen 23, Número 2 (1990), páginas 5–16.

[Eisenberg and Melton 1999] A. Eisenberg and J. Melton, «SQL:1999, formerly known as SQL3», *ACM SIGMOD Record*, Volumen 28, Número 1 (1999).

[Eisenberg and Melton 2004a] A. Eisenberg and J. Melton, «Advancements in SQL/XML», *ACM SIGMOD Record*, Volumen 33, Número 3 (2004), páginas 79–86.

[Eisenberg and Melton 2004b] A. Eisenberg and J. Melton, «An Early Look at XQuery API for Java (XQJ)», *ACM SIGMOD Record*, Volumen 33, Número 2 (2004), páginas 105–111.

[Eisenberg et ál. 2004] A. Eisenberg, J. Melton, K. G. Kulkarni, J.-E. Michels y F. Zemke, «SQL:2003 Has Been Published», *ACM SIGMOD Record*, Volumen 33, Número 1 (2004), páginas 119–126.

[Elhemali et ál. 2007] M. Elhemali, C. A. Galindo-Legaria, T. Grabs y M. Joshi, «Execution strategies for SQL subqueries», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2007), páginas 993–1004.

[Ellis 1987] C. S. Ellis, «Concurrency in Linear Hashing», *ACM Transactions on Database Systems*, Volumen 12, Número 2 (1987), páginas 195–217.

[Elmasri and Navathe 2006] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 5th edition, Addison Wesley (2006).

[Epstein et ál. 1978] R. Epstein, M. R. Stonebraker y E. Wong, «Distributed Query Processing in a Relational Database System», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1978), páginas 169–180.

[Escobar-Molano et ál. 1993] M. Escobar-Molano, R. Hull y D. Jacobs, «Safety and Translation of Calculus Queries with Scalar Functions», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993), páginas 253–264.

[Eswaran et ál. 1976] K. P. Eswaran, J. N. Gray, R. A. Lorie y I. L. Traiger, «The Notions of Consistency and Predicate Locks in a Database System», *Communications of the ACM*, Volumen 19, Número 11 (1976), páginas 624–633.

[Fagin 1977] R. Fagin, «Multivalued Dependencies and a New Normal Form for Relational Databases», *ACM Transactions on Database Systems*, Volumen 2, Número 3 (1977), páginas 262–278.

[Fagin 1979] R. Fagin, «Normal Forms and Relational Database Operators», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1979), páginas 153–160.

[Fagin 1981] R. Fagin, «A Normal Form for Relational Databases That Is Based on Domains and Keys», *ACM Transactions on Database Systems*, Volumen 6, Número 3 (1981), páginas 387–415.

[Fagin et ál. 1979] R. Fagin, J. Nievergelt, N. Pippenger y H. R. Strong, «Extendible Hashing — A Fast Access Method for Dynamic Files», *ACM Transactions on Database Systems*, Volumen 4, Número 3 (1979), páginas 315–344.

[Faloutsos and Lin 1995] C. Faloutsos and K.-I. Lin, «Fast Map: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 163–174.

[Fayyad et ál. 1995] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth y R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, MIT Press (1995).

[Fekete et ál. 2005] A. Fekete, D. Liarokapis, E. O'Neil, P. O'Neil y D. Shasha, «Making Snapshot Isolation Serializable», *ACM Transactions on Database Systems*, Volumen 30, Número 2 (2005).

[Finkel and Bentley 1974] R. A. Finkel and J. L. Bentley, «Quad Trees: A Data Structure for Retrieval on Composite Keys», *Acta Informatica*, Volumen 4, (1974), páginas 1–9.

[Fischer 2006] L. Fischer, editor, *Workflow Handbook 2001*, Future Strategies (2006).

[Florescu and Kossmann 1999] D. Florescu and D. Kossmann, «Storing and Querying XML Data Using an RDBMS», *IEEE Data Engineering Bulletin (Special Issue on XML)* (1999), páginas 27–35.

[Florescu et ál. 2000] D. Florescu, D. Kossmann y I. Monalescu, «Integrating Keyword Search into XML Query Processing», *Proc. of the International World Wide Web Conf.* (2000), páginas 119–135. Also appears in *Computer Networks*, Vol. 33, páginas 119-135.

[Fredkin 1960] E. Fredkin, «Trie Memory», *Communications of the ACM*, Volumen 4, Número 2 (1960), páginas 490–499.

[Freedman and DeWitt 1995] C. S. Freedman and D. J. DeWitt, «The SPIFFI Scalable Video-on-Demand Server», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 352–363.

[Funderburk et ál. 2002a] J. E. Funderburk, G. Kiernan, J. Shanmugasundaram, E. Shekita y C. Wei, «XTABLES: Bridging Relational Technology and XML», *IBM Systems Journal*, Volumen 41, Número 4 (2002), páginas 616–641.

[Funderburk et ál. 2002b] J. E. Funderburk, S. Malaika y B. Reinwald, «XML Programming with SQL/XML and XQuery», *IBM Systems Journal*, Volumen 41, Número 4 (2002), páginas 642–665.

[Galindo-Legaria 1994] C. Galindo-Legaria, «Outerjoins as Disjunctions», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1994).

[Galindo-Legaria and Joshi 2001] C. A. Galindo-Legaria and M. M. Joshi, «Orthogonal Optimization of Subqueries and Aggregation», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2001).

[Galindo-Legaria and Rosenthal 1992] C. Galindo-Legaria and A. Rosenthal, «How to Extend a Conventional Optimizer to Handle One and Two-Sided Outer join», *Proc. of the International Conf. on Data Engineering* (1992), páginas 402–409.

[Galindo-Legaria et ál. 2004] C. Galindo-Legaria, S. Stefani y F. Waas, «Query Processing for SQL Updates», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2004), páginas 844–849.

[Ganguly 1998] S. Ganguly, «Design and Analysis of Parametric Query Optimization Algorithms», *Proc. of the International Conf. on Very Large Databases* (1998).

[Ganguly et ál. 1992] S. Ganguly, W. Hasan y R. Krishnamurthy, «Query Optimization for Parallel Execution», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1992).

[Ganguly et ál. 1996] S. Ganguly, P. Gibbons, Y. Matias y A. Silberschatz, «A Sampling Algorithm for Estimating Join Size», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Ganski and Wong 1987] R. A. Ganski and H. K. T. Wong, «Optimization of Nested SQL Queries Revisited», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987).

[García and Korth 2005] P. García and H. F. Korth, «Multithreaded Architectures and the Sort Benchmark», *Proc. of the First International Workshop on Data Management on Modern Hardward (DaMoN)* (2005).

[García-Molina 1982] H. García-Molina, «Elections in Distributed Computing Systems», *IEEE Transactions on Computers*, Volumen C-31, Número 1 (1982), páginas 48–59.

[García-Molina and Salem 1987] H. García-Molina and K. Salem, «Sagas», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987), páginas 249–259.

[García-Molina and Salem 1992] H. García-Molina and K. Salem, «Main Memory Database Systems: An Overview», *IEEE Transactions on Knowledge and Data Engineering*, Volumen 4, Número 6 (1992), páginas 509–516.

[García-Molina et ál. 2008] H. García-Molina, J. D. Ullman y J. D. Widom, *Database Systems: The Complete Book*, 2nd edition, Prentice Hall (2008).

[Georgakopoulos et ál. 1994] D. Georgakopoulos, M. Rusinkiewicz y A. Seth, «Using Tickets to Enforce the Serializability of Multidatabase Transactions», *IEEE Transactions on Knowledge and Data Engineering*, Volumen 6, Número 1 (1994), páginas 166–180.

[Gilbert and Lynch 2002] S. Gilbert and N. Lynch, «Brewer 's conjecture and the feasibility of consistent, available, partition-tolerant web services», *SIGACT News*, Volumen 33, Número 2 (2002), páginas 51–59.

[Graefe 1990] G. Graefe, «Encapsulation of Parallelism in the Volcano Query Processing System», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990), páginas 102–111.

[Graefe 1995] G. Graefe, «The Cascades Framework for Query Optimization», *Data Engineering Bulletin*, Volumen 18, Número 3 (1995), páginas 19–29.

[Graefe 2008] G. Graefe, «The Five-Minute Rule 20 Years Later: and How Flash Memory Changes the Rules», *ACM Queue*, Volumen 6, Número 4 (2008), páginas 40–52.

[Graefe and McKenna 1993a] G. Graefe and W. McKenna, «The Volcano Optimizer Generator», *Proc. of the International Conf. on Data Engineering* (1993), páginas 209–218.

[Graefe and McKenna 1993b] G. Graefe and W. J. McKenna, «Extensibility and Search Efficiency in the Volcano Optimizer Generator», *Proc. of the International Conf. on Data Engineering* (1993).

[Graefe et ál. 1998] G. Graefe, R. Bunker y S. Cooper, «Hash Joins and Hash Teams in Microsoft SQL Server», *Proc. of the International Conf. on Very Large Databases* (1998), páginas 86–97.

[Gray 1978] J. Gray. «Notes on Data Base Operating System», Bayer et ál. [1978], páginas 393–481 (1978).

[Gray 1981] J. Gray, «The Transaction Concept: Virtues and Limitations», *Proc. of the International Conf. on Very Large Databases* (1981), páginas 144–154.

[Gray 1991] J. Gray, *The Benchmark Handbook for Database and Transaction Processing Systems*, 2nd edition, Morgan Kaufmann (1991).

[Gray and Graefe 1997] J. Gray and G. Graefe, «The Five-Minute Rule Ten Years Later y Other Computer Storage Rules of Thumb», *SIGMOD Record*, Volumen 26, Número 4 (1997), páginas 63–68.

[Gray and Reuter 1993] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann (1993).

[Gray et ál. 1975] J. Gray, R. A. Lorie y G. R. Putzolu, «Granularity of Locks and Degrees of Consistency in a Shared Data Base», *Proc. of the International Conf. on Very Large Databases* (1975), páginas 428–451.

[Gray et ál. 1976] J. Gray, R. A. Lorie, G. R. Putzolu y I. L. Traiger, *Granularity of Locks and Degrees of Consistency in a Shared Data Base*, Nijssen (1976).

[Gray et ál. 1981] J. Gray, P. R. McJones y M. Blasgen, «The Recovery Manager of the System R Database Manager», *ACM Computing Survey*, Volumen 13, Number 2 (1981), páginas 223–242.

[Gray et ál. 1995] J. Gray, A. Bosworth, A. Layman y H. Pirahesh, «Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals», Technical report, Microsoft Research (1995).

[Gray et ál. 1996] J. Gray, P. Helland y P. O'Neil, «The Dangers of Replication and a Solution», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 173–182.

[Gray et ál. 1997] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow y H. Pirahesh, «Data Cube: A Relational Aggregation Operator Generalizing Groupby, Cross-Tab, and Sub Totals», *Data Mining and Knowledge Discovery*, Volumen 1, Número 1 (1997), páginas 29–53.

[Gregersen and Jensen 1999] H. Gregersen and C. S. Jensen, «Temporal Entity-Relationship Models-A Survey», *IEEE Transactions on Knowledge and Data Engineering*, Volumen 11, Número 3 (1999), páginas 464–497.

[Grossman and Frieder 2004] D. A. Grossman and O. Frieder, *Information Retrieval: Algorithms and Heuristics*, 2nd edition, Springer Verlag (2004).

[Gunning 2008] P. K. Gunning, *DB2 9 for Developers*, MC Press (2008).

[Guo et ál. 2003] L. Guo, F. Shao, C. Botev y J. Shanmugasundaram, «XRANK: Ranked Keyword Search over XML Documents», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2003).

[Guttman 1984] A. Guttman, «R-Trees: A Dynamic Index Structure for Spatial Searching», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1984), páginas 47–57.

[Haas et ál. 1989] L. M. Haas, J. C. Freytag, G. M. Lohman y H. Pirahesh, «Extensible Query Processing in Starburst», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1989), páginas 377–388.

[Haas et ál. 1990] L. M. Haas, W. Chang, G. M. Lohman, J. McPherson, P. F. Wilms, G. Lapis, B. G. Lindsay, H. Pirahesh, M. J. Carey y E. J. Shekita, «Starburst Mid-Flight: As the Dust Clears», *IEEE Transactions on Knowledge and Data Engineering*, Volumen 2, Número 1 (1990), páginas 143–160.

[Haerder and Reuter 1983] T. Haerder and A. Reuter, «Principles of Transaction Oriented Database Recovery», *ACM Computing Survey*, Volumen 15, Número 4 (1983), páginas 287–318.

[Haerder and Rothermel 1987] T. Haerder and K. Rothermel, «Concepts for Transaction Recovery in Nested Transactions», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987), páginas 239–248.

[Halsall 2006] F. Halsall, *Computer Networking and the Internet: With Internet and Multiamedia Applications*, Addison Wesley (2006).

[Han and Kamber 2000] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann (2000).

[Harinarayan et ál. 1996] V. Harinarayan, J. D. Ullman y A. Rajaraman, «Implementing Data Cubes Efficiently», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Haritsa et ál. 1990] J. Haritsa, M. Carey y M. Livny, «On Being Optimistic about Real-Time Constraints», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990).

[Harizopoulos and Ailamaki 2004] S. Harizopoulos and A. Ailamaki, «STEPS towards Cache-resident Transaction Processing», *Proc. of the International Conf. on Very Large Databases* (2004), páginas 660–671.

[Hellerstein and Stonebraker 2005] J. M. Hellerstein and M. Stonebraker, editors, *Readings in Database Systems*, 4th edition, Morgan Kaufmann (2005).

[Hellerstein et ál. 1995] J. M. Hellerstein, J. F. Naughton y A. Pfeffer, «Generalized Search Trees for Database Systems», *Proc. of the International Conf. on Very Large Databases* (1995), páginas 562–573.

[Hennessy et ál. 2006] J. L. Hennessy, D. A. Patterson y D. Goldberg, *Computer Architecture: A Quantitative Approach*, 4th edition, Morgan Kaufmann (2006).

[Hevner and Yao 1979] A. R. Hevner and S. B. Yao, «Query Processing in Distributed Database Systems», *IEEE Transactions on Software Engineering*, Volumen SE-5, Número 3 (1979), páginas 177–187.

[Heywood et ál. 2002] I. Heywood, S. Cornelius y S. Carver, *An Introduction to Geographical Information Systems*, 2nd edition, Prentice Hall (2002).

[Hong et ál. 1993] D. Hong, T. Johnson y S. Chakravarthy, «Real-Time Transaction Scheduling: A Cost Conscious Approach», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993).

[Howes et ál. 1999] T. A. Howes, M. C. Smith y G. S. Good, *Understanding and Deploying LDAP Directory Services*, Macmillan Publishing (1999).

[Hristidis and Papakonstantinou 2002] V. Hristidis and Y. Papakonstantinou, «DISCOVER: Keyword Search in Relational Databases», *Proc. of the International Conf. on Very Large Databases* (2002).

[Huang and García-Molina 2001] Y. Huang and H. García-Molina, «Exactly-once Semantics in a Replicated Messaging System», *Proc. of the International Conf. on Data Engineering* (2001), páginas 3–12.

[Hulgeri and Sudarshan 2003] A. Hulgeri and S. Sudarshan, «AniPQO: Almost Non-Intrusive Parametric Query Optimization for Non-Linear Cost Functions», *Proc. of the International Conf. on Very Large Databases* (2003).

[IBM 1987] IBM, «Systems Application Architecture: Common Programming Interface, Database Reference», Informe técnico, IBM Corporation, IBM Form Número SC26–4348–0 (1987).

[Ilyas et ál. 2008] I. Ilyas, G. Beskales y M. A. Soliman, «A Survey of top-*k* query processing techniques in relational database systems», *ACM Computing Surveys*, Volumen 40, Número 4 (2008).

[Imielinski and Badrinath 1994] T. Imielinski and B. R. Badrinath, «Mobile Computing — Solutions and Challenges», *Communications of the ACM*, Volumen 37, Número 10 (1994).

[Imielinski and Korth 1996] T. Imielinski and H. F. Korth, editors, *Mobile Computing*, Kluwer Academic Publishers (1996).

[Ioannidis and Christodoulakis 1993] Y. Ioannidis and S. Christodoulakis, «Optimal Histograms for Limiting Worst-Case Error Propagation in the Size of Join Results», *ACM Transactions on Database Systems*, Volumen 18, Número 4 (1993), páginas 709–748.

[Ioannidis and Poosala 1995] Y. E. Ioannidis and V. Poosala, «Balancing Histogram Optimality and Practicality for Query Result Size Estimation», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 233–244.

[Ioannidis et ál. 1992] Y. E. Ioannidis, R. T. Ng, K. Shim y T. K. Sellis, «Parametric Query Optimization», *Proc. of the International Conf. on Very Large Databases* (1992), páginas 103–114.

[Jackson and Moulinier 2002] P. Jackson and I. Moulinier, *Natural Language Processing for Online Applications: Text Retrieval, Extraction, and Categorization*, John Benjamin (2002).

[Jagadish et ál. 1993] H. V. Jagadish, A. Silberschatz y S. Sudarshan, «Recovering from Main-Memory Lapses», *Proc. of the International Conf. on Very Large Databases* (1993).

[Jagadish et ál. 1994] H. Jagadish, D. Lieuwen, R. Rastogi, A. Silberschatz y S. Sudarshan, «Dali: A High Performance Main Memory Storage Manager», *Proc. of the International Conf. on Very Large Databases* (1994).

[Jain and Dubes 1988] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall (1988).

[Jensen et ál. 1994] C. S. Jensen et ál., «A Consensus Glossary of Temporal Database Concepts», *ACM SIGMOD Record*, Volumen 23, Número 1 (1994), páginas 52–64.

[Jensen et ál. 1996] C. S. Jensen, R. T. Snodgrass y M. Soo, «Extending Existing Dependency Theory to Temporal Databases», *IEEE Transactions on Knowledge and Data Engineering*, Volumen 8, Número 4 (1996), páginas 563–582.

[Johnson 1999] T. Johnson, «Performance Measurements of Compressed Bitmap Indices», *Proc. of the International Conf. on Very Large Databases* (1999).

[Johnson and Shasha 1993] T. Johnson and D. Shasha, «The Performance of Concurrent B-Tree Algorithms», *ACM Transactions on Database Systems*, Volumen 18, Número 1 (1993).

[Jones and Willet 1997] K. S. Jones and P. Willet, editors, *Readings in Information Retrieval*, Morgan Kaufmann (1997).

[Jordan and Russell 2003] D. Jordan and C. Russell, *Java Data Objects*, O'Reilly (2003).

[Jorwekar et ál. 2007] S. Jorwekar, A. Fekete, K. Ramamritham y S. Sudarshan, «Automating the Detection of Snapshot Isolation Anomalies», *Proc. of the International Conf. on Very Large Databases* (2007), páginas 1263–1274.

[Joshi 1991] A. Joshi, «Adaptive Locking Strategies in a Multi-Node Shared Data Model Environment», *Proc. of the International Conf. on Very Large Databases* (1991).

[Kanne and Moerkotte 2000] C.-C. Kanne and G. Moerkotte, «Efficient Storage of XML Data», *Proc. of the International Conf. on Data Engineering* (2000), página 198.

[Katz et ál. 2004] H. Katz, D. Chamberlin, D. Draper, M. Fernández, M. Kay, J. Robie, M. Rys, J. Simeon, J. Tivy y P. Wadler, *XQuery from the Experts: A Guide to the W3C XML Query Language*, Addison Wesley (2004).

[Kaushik et ál. 2004] R. Kaushik, R. Krishnamurthy, J. F. Naughton y R. Ramakrishnan, «On the Integration of Structure Indexes and Inverted Lists», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2004).

[Kedem and Silberschatz 1979] Z. M. Kedem and A. Silberschatz, «Controlling Concurrency Using Locking Protocols», *Proc. of the Annual IEEE Symposium on Foundations of Computer Science* (1979), páginas 275–285.

[Kedem and Silberschatz 1983] Z. M. Kedem and A. Silberschatz, «Locking Protocols: From Exclusive to Shared Locks», *Journal of the ACM*, Volumen 30, Número 4 (1983), páginas 787–804.

[Kifer et ál. 2005] M. Kifer, A. Bernstein y P. Lewis, *Database Systems: An Application Oriented Approach, Complete Version*, 2nd edition, Addison Wesley (2005).

[Kim 1982] W. Kim, «On Optimizing an SQL-like Nested Query», *ACM Transactions on Database Systems*, Volumen 3, Número 3 (1982), páginas 443–469.

[Kim 1995] W. Kim, editor, *Modern Database Systems*, ACM Press (1995).

[King et ál. 1991] R. P. King, N. Halim, H. García-Molina y C. Polyzois, «Management of a Remote Backup Copy for Disaster Recovery», *ACM Transactions on Database Systems*, Volumen 16, Número 2 (1991), páginas 338–368.

[Kitsuregawa and Ogawa 1990] M. Kitsuregawa and Y. Ogawa, «Bucket Spreading Parallel Hash: A New, Robust, Parallel Hash Join Method for Skew in the Super Database Computer», *Proc. of the International Conf. on Very Large Databases* (1990), páginas 210–221.

[Kleinberg 1999] J. M. Kleinberg, «Authoritative Sources in a Hyperlinked Environment», *Journal of the ACM*, Volumen 46, Número 5 (1999), páginas 604–632.

[Kleinrock 1975] L. Kleinrock, *Queueing Systems*, Wiley-Interscience (1975).

[Klug 1982] A. Klug, «Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions», *Journal of the ACM*, Volumen 29, Número 3 (1982), páginas 699–717.

[Knapp 1987] E. Knapp, «Deadlock Detection in Distributed Databases», *ACM Computing Survey*, Volumen 19, Número 4 (1987).

[Knuth 1973] D. E. Knuth, *The Art of Computer Programming, Volumen 3*, Addison Wesley, Sorting and Searching (1973).

[Kohavi and Provost 2001] R. Kohavi and F. Provost, editors, *Applications of Data Mining to Electronic Commerce*, Kluwer Academic Publishers (2001).

[Konstan et ál. 1997] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon y J. Riedl, «GroupLens: Applying Collaborative Filtering to Usenet News», *Communications of the ACM*, Volumen 40, Número 3 (1997), páginas 77–87.

[Korth 1982] H. F. Korth, «Deadlock Freedom Using Edge Locks», *ACM Transactions on Database Systems*, Volumen 7, Número 4 (1982), páginas 632–652.

[Korth 1983] H. F. Korth, «Locking Primitives in a Database System», *Journal of the ACM*, Volumen 30, Número 1 (1983), páginas 55–79.

[Korth and Speegle 1990] H. F. Korth and G. Speegle, «Long Duration Transactions in Software Design Projects», *Proc. of the International Conf. on Data Engineering* (1990), páginas 568–575.

[Korth and Speegle 1994] H. F. Korth and G. Speegle, «Formal Aspects of Concurrency Control in Long Duration Transaction Systems Using the NT/PV Model», *ACM Transactions on Database Systems*, Volumen 19, Número 3 (1994), páginas 492–535.

[Krishnaprasad et ál. 2004] M. Krishnaprasad, Z. Liu, A. Manikutty, J. W. Warner, V. Arora y S. Kotsovolos, «Query Rewrite for XML in Oracle XML DB», *Proc. of the International Conf. on Very Large Databases* (2004), páginas 1122–1133.

[Kung and Lehman 1980] H. T. Kung and P. L. Lehman, «Concurrent Manipulation of Binary Search Trees», *ACM Transactions on Database Systems*, Volumen 5, Number 3 (1980), páginas 339–353.

[Kung and Robinson 1981] H. T. Kung and J. T. Robinson, «Optimistic Concurrency Control», *ACM Transactions on Database Systems*, Volumen 6, Número 2 (1981), páginas 312–326.

[Kurose and Ross 2005] J. Kurose and K. Ross, *Computer Networking — A Top-Down Approach Featuring the Internet*, 3rd edition, Addison Wesley (2005).

[Lahiri et ál. 2001] T. Lahiri, A. Ganesh, R. Weiss y A. Joshi, «Fast-Start: Quick Fault Recovery in Oracle», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2001).

[Lam and Kuo 2001] K.-Y. Lam and T.-W. Kuo, editors, *Real-Time Database Systems*, Kluwer Academic Publishers (2001).

[Lamb et ál. 1991] C. Lamb, G. Landis, J. Orenstein y D. Weinreb, «The Object-Store Database System», *Communications of the ACM*, Volumen 34, Número 10 (1991), páginas 51–63.

[Lamport 1978] L. Lamport, «Time, Clocks y the Ordering of Events in a Distributed System», *Communications of the ACM*, Volumen 21, Número 7 (1978), páginas 558–565.

[Lampson and Sturgis 1976] B. Lampson and H. Sturgis, «Crash Recovery in a Distributed Data Storage System», Technical report, Computer Science Laboratory, Xerox Palo Alto Research Center,Palo Alto (1976).

[Lecluse et ál. 1988] C. Lecluse, P. Richard y F. Velez, «O2: An Object-Oriented Data Model», *Proc. of the International Conf. on Very Large Databases* (1988), páginas 424–433.

[Lehman and Yao 1981] P. L. Lehman and S. B. Yao, «Efficient Locking for Concurrent Operations on B-trees», *ACM Transactions on Database Systems*, Volumen 6, Número 4 (1981), páginas 650–670.

[Lehner et ál. 2000] W. Lehner, R. Sidle, H. Pirahesh y R. Cochrane, «Maintenance of Automatic Summary Tables», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2000), páginas 512–513.

[Lindsay et ál. 1980] B. G. Lindsay, P. G. Selinger, C. Galtieri, J. N. Gray, R. A. Lorie, T. G. Price, G. R. Putzolu, I. L. Traiger y B. W. Wade. «Notes on Distributed Databases», Draffen and Poole, editors, *Distributed Data Bases*, páginas 247–284. Cambridge University Press (1980).

[Litwin 1978] W. Litwin, «Virtual Hashing: A Dynamically Changing Hashing», In *Proc. of the International Conf. on Very Large Databases* (1978), páginas 517–523.

[Litwin 1980] W. Litwin, «Linear Hashing: A New Tool for File and Table Addressing», *Proc. of the International Conf. on Very Large Databases* (1980), páginas 212–223.

[Litwin 1981] W. Litwin, «Trie Hashing», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1981), páginas 19–29.

[Lo and Ravishankar 1996] M.-L. Lo and C. V. Ravishankar, «Spatial Hash-Joins», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Loeb 1998] L. Loeb, *Secure Electronic Transactions: Introduction and Technical Reference*, Artech House (1998).

[Lomet 1981] D. G. Lomet, «Digital B-trees», *Proc. of the International Conf. on Very Large Databases* (1981), páginas 333–344.

[Lomet et ál. 2009] D. Lomet, A. Fekete, G. Weikum y M. Zwilling, «Unbundling Transaction Services in the Cloud», *Proc. 4th Biennial Conference on Innovative Data Systems Research* (2009).

[Lu et ál. 1991] H. Lu, M. Shan y K. Tan, «Optimization of Multi-Way Join Queries for Parallel Execution», *Proc. of the International Conf. on Very Large Databases* (1991), páginas 549–560.

[Lynch and Merritt 1986] N. A. Lynch and M. Merritt, «Introduction to the Theory of Nested Transactions», *Proc. of the International Conf. on Database Theory* (1986).

[Lynch et ál. 1988] N. A. Lynch, M. Merritt, W. Weihl y A. Fekete, «A Theory of Atomic Transactions», *Proc. of the International Conf. on Database Theory* (1988), páginas 41–71.

[Maier 1983] D. Maier, *The Theory of Relational Databases*, Computer Science Press (1983).

[Manning et ál. 2008] C. D. Manning, P. Raghavan y H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press (2008).

[Martin et ál. 1989] J. Martin, K. K. Chapman y J. Leben, *DB2, Concepts, Design, and Programming*, Prentice Hall (1989).

[Mattison 1996] R. Mattison, *Data Warehousing: Strategies, Technologies, and Techniques*, McGraw-Hill (1996).

[McHugh and Widom 1999] J. McHugh and J. Widom, «Query Optimization for XML», *Proc. of the International Conf. on Very Large Databases* (1999), páginas 315–326.

[Mehrotra et ál. 1991] S. Mehrotra, R. Rastogi, H. F. Korth y A. Silberschatz, «Non-Serializable Executions in Heterogeneous Distributed Database Systems», *Proc. of the International Conf. on Parallel and Distributed Information Systems* (1991).

[Mehrotra et ál. 2001] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth y A. Silberschatz, «Overcoming Heterogeneity and Autonomy in Multidatabase Systems.», *Inf. Comput.*, Volumen 167, Número 2 (2001), páginas 137–172.

[Melnik et ál. 2007] S. Melnik, A. Adya y P. A. Bernstein, «Compiling mappings to bridge applications and databases», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2007), páginas 461–472.

[Melton 2002] J. Melton, *Advanced SQL:1999 – Understanding Object-Relational and Other Advanced Features*, Morgan Kaufmann (2002).

[Melton and Eisenberg 2000] J. Melton and A. Eisenberg, *Understanding SQL and Java Together : A Guide to SQLJ, JDBC, and Related Technologies*, Morgan Kaufmann (2000).

[Melton and Simon 1993] J. Melton and A. R. Simon, *Understanding The New SQL: A Complete Guide*, Morgan Kaufmann (1993).

[Melton and Simon 2001] J. Melton and A. R. Simon, *SQL:1999, Understanding Relational Language Components*, Morgan Kaufmann (2001).

[Microsoft 1997] Microsoft, *Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference*, Microsoft Press (1997).

[Mistry et ál. 2001] H. Mistry, P. Roy, S. Sudarshan y K. Ramamritham, «Materialized View Selection and Maintenance Using Multi-Query Optimization», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2001).

[Mitchell 1997] T. M. Mitchell, *Machine Learning*, McGraw-Hill (1997).

[Mohan 1990a] C. Mohan, «ARIES/KVL: A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operations on B-Tree indexes», *Proc. of the International Conf. on Very Large Databases* (1990), páginas 392–405.

[Mohan 1990b] C. Mohan, «Commit-LSN: A Novel and Simple Method for Reducing Locking and Latching in Transaction Processing Systems», *Proc. of the International Conf. on Very Large Databases* (1990), páginas 406–418.

[Mohan 1993] C. Mohan, «IBM's Relational Database Products:-Features and Technologies», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1993).

[Mohan and Levine 1992] C. Mohan and F. Levine, «ARIES/IM:An Efficient and High-Concurrency Index Management Method Using Write-Ahead Logging», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1992).

[Mohan and Lindsay 1983] C. Mohan and B. Lindsay, «Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions», *Proc. of the ACM Symposium on Principles of Distributed Computing* (1983).

[Mohan and Narang 1992] C. Mohan and I. Narang, «Efficient Locking and Caching of Data in the Multisystem Shared Disks Transaction Environment», *Proc. of the International Conf. on Extending Database Technology* (1992).

[Mohan and Narang 1994] C. Mohan and I. Narang, «ARIES/CSA: A Method for Database Recovery in Client-Server Architectures», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1994), páginas 55–66.

[Mohan et ál. 1986] C. Mohan, B. Lindsay y R. Obermarck, «Transaction Management in the R* Distributed Database Management System», *ACM Transactions on Database Systems*, Volumen 11, Número 4 (1986), páginas 378–396.

[Mohan et ál. 1992] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh y P. Schwarz, «ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging», *ACM Transactions on Database Systems*, Volumen 17, Número 1 (1992).

[Moss 1985] J. E. B. Moss, *Nested Transactions: An Approach to Reliable Distributed Computing*, MIT Press (1985).

[Moss 1987] J. E. B. Moss, «Log-Based Recovery for Nested Transactions», *Proc. of the International Conf. on Very Large Databases* (1987), páginas 427–432.

[Murthy and Banerjee 2003] R. Murthy and S. Banerjee, «XML Schemas in Oracle XML DB», *Proc. of the International Conf. on Very Large Databases* (2003), páginas 1009–1018.

[Nakayama et ál. 1984] T. Nakayama, M. Hirakawa y T. Ichikawa, «Architecture and Algorithm for Parallel Execution of a Join Operation», *Proc. of the International Conf. on Data Engineering* (1984).

[Ng and Han 1994] R. T. Ng and J. Han, «Efficient and Effective Clustering Methods for Spatial Data Mining», *Proc. of the International Conf. on Very Large Databases* (1994).

[NIST 1993] NIST, «Integration Definition for Information Modeling (IDEF1X)», Technical Report Federal Information Processing Standards Publication 184, National Institute of Standards and Technology (NIST), Available at www.idef.com/Downloads/pdf/Idef1x.pdf (1993).

[Nyberg et ál. 1995] C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray y D. B. Lomet, «AlphaSort: A Cache-Sensitive Parallel External Sort», *VLDB Journal*, Volumen 4, Número 4 (1995), páginas 603–627.

[O'Neil and O'Neil 2000] P. O'Neil and E. O'Neil, *Database: Principles, Programming, Performance*, 2nd edition, Morgan Kaufmann (2000).

[O'Neil and Quass 1997] P. O'Neil and D. Quass, «Improved Query Performance with Variant Indexes», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1997).

[O'Neil et ál. 2004] P. E. O'Neil, E. J. O'Neil, S. Pal, I. Cseri, G. Schaller y N. Westbury, «ORDPATHs: Insert-Friendly XML Node Labels», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2004), páginas 903–908.

[Ooi and S. Parthasarathy 2009] B. C. Ooi and e. S. Parthasarathy, «Special Issue on Data Management on Cloud Computing Platforms», *Data Engineering Bulletin*, Volumen 32, Número 1 (2009).

[Orenstein 1982] J. A. Orenstein, «Multidimensional Tries Used for Associative Searching», *Information Processing Letters*, Volumen 14, Número 4 (1982), páginas 150–157.

[Ozcan et ál. 1997] F. Ozcan, S. Nural, P. Koksal, C. Evrendilek y A. Dogac, «Dynamic Query Optimization in Multidatabases», *Data Engineering Bulletin*, Volumen 20, Número 3 (1997), páginas 38–45.

[Ozden et ál. 1994] B. Ozden, A. Biliris, R. Rastogi y A. Silberschatz, «A Lowcost Storage Server for a Movie on Demand Database», *Proc. of the International Conf. on Very Large Databases* (1994).

[Ozden et ál. 1996a] B. Ozden, R. Rastogi, P. Shenoy y A. Silberschatz, «Fault Tolerant Architectures for Continuous Media Servers», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Ozden et ál. 1996b] B. Ozden, R. Rastogi y A. Silberschatz, «On the Design of a Low-Cost Video-on-Demand Storage System», *Multimedia Systems Journal*, Volumen 4, Número 1 (1996), páginas 40–54.

[Ozsoyoglu and Snodgrass 1995] G. Ozsoyoglu and R. Snodgrass, «Temporal and Real-Time Databases: A Survey», *IEEE Transactions on Knowledge and Data Engineering*, Volumen 7, Número 4 (1995), páginas 513–532.

[Ozsu and Valduriez 1999] T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, 2nd edition, Prentice Hall (1999).

[Padmanabhan et ál. 2003] S. Padmanabhan, B. Bhattacharjee, T. Malkemus, L. Cranston y M. Huras, «Multi-Dimensional Clustering: A New Data Layout Scheme in DB2», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2003), páginas 637–641.

[Pal et ál. 2004] S. Pal, I. Cseri, G. Schaller, O. Seeliger, L. Giakoumakis y V. Zolotov, «Indexing XML Data Stored in a Relational Database», *Proc. of the International Conf. on Very Large Databases* (2004), páginas 1134–1145.

[Pang et ál. 1995] H.-H. Pang, M. J. Carey y M. Livny, «Multiclass Scheduling in Real-Time Database Systems», *IEEE Transactions on Knowledge and Data Engineering*, Volumen 2, Número 4 (1995), páginas 533–551.

[Papakonstantinou et ál. 1996] Y. Papakonstantinou, A. Gupta, and L. Haas, «Capabilities-Based Query Rewriting in Mediator Systems», *Proc. of the International Conf. on Parallel and Distributed Information Systems* (1996).

[Parker et ál. 1983] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser y C. Kline, «Detection of Mutual Inconsistency in Distributed Systems», *IEEE Transactions on Software Engineering*, Volumen 9, Número 3 (1983), páginas 240–246.

[Patel and DeWitt 1996] J. Patel and D. J. DeWitt, «Partition Based Spatial-Merge Join», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Patterson 2004] D. P. Patterson, «Latency Lags Bandwidth», *Communications of the ACM*, Volumen 47, Número 10 (2004), páginas 71–75.

[Patterson et ál. 1988] D. A. Patterson, G. Gibson y R. H. Katz, «A Case for Redundant Arrays of Inexpensive Disks (RAID)», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1988), páginas 109–116.

[Pellenkoft et ál. 1997] A. Pellenkoft, C. A. Galindo-Legaria y M. Kersten, «The Complexity of Transformation-Based Join Enumeration», *Proc. of the International Conf. on Very Large Databases* (1997), páginas 306–315.

[Peterson and Davie 2007] L. L. Peterson and B. S. Davie, *Computer Networks: a Systems Approach*, Morgan Kaufmann Publishers Inc. (2007).

[Pless 1998] V. Pless, *Introduction to the Theory of Error-Correcting Codes*, 3rd edition, John Wiley and Sons (1998).

[Poe 1995] V. Poe, *Building a Data Warehouse for Decision Support*, Prentice Hall (1995).

[Polyzois and García-Molina 1994] C. Polyzois and H. García-Molina, «Evaluation of Remote Backup Algorithms for Transaction-Processing Systems», *ACM Transactions on Database Systems*, Volumen 19, Número 3 (1994), páginas 423–449.

[Poosala et ál. 1996] V. Poosala, Y. E. Ioannidis, P. J. Haas y E. J. Shekita, «Improved Histograms for Selectivity Estimation of Range Predicates», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 294–305.

[Popek et ál. 1981] G. J. Popek, B. J. Walker, J. M. Chow, D. Edwards, C. Kline, G. Rudisin y G. Thiel, «LOCUS: A Network Transparent, High Reliability Distributed System», *Proc. of the Eighth Symposium on Operating System Principles* (1981), páginas 169–177.

[Pöss and Potapov 2003] M. Pöss and D. Potapov, «Data Compression in Oracle», *Proc. of the International Conf. on Very Large Databases* (2003), páginas 937–947.

[Rahm 1993] E. Rahm, «Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems», *ACM Transactions on Database Systems*, Volumen 8, Número 2 (1993).

[Ramakrishna and Larson 1989] M. V. Ramakrishna and P. Larson, «File Organization Using Composite Perfect Hashing», *ACM Transactions on Database Systems*, Volumen 14, Número 2 (1989), páginas 231–263.

[Ramakrishnan and Gehrke 2002] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd edition, McGraw-Hill (2002).

[Ramakrishnan and Ullman 1995] R. Ramakrishnan and J. D. Ullman, «A Survey of Deductive Database Systems», *Journal of Logic Programming*, Volumen 23, Número 2 (1995), páginas 125–149.

[Ramakrishnan et ál. 1992] R. Ramakrishnan, D. Srivastava y S. Sudarshan, *Controlling the Search in Bottom-up Evaluation* (1992).

[Ramesh et ál. 1989] R. Ramesh, A. J. G. Babu y J. P. Kincaid, «Index Optimization: Theory and Experimental Results», *ACM Transactions on Database Systems*, Volumen 14, Número 1 (1989), páginas 41–74.

[Rangan et ál. 1992] P. V. Rangan, H. M. Vin y S. Ramanathan, «Designing an On-Demand Multimedia Service», *Communications Magazine*, Volumen 1, Número 1 (1992), páginas 56–64.

[Rao and Ross 2000] J. Rao and K. A. Ross, «Making B+-Trees Cache Conscious in Main Memory», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2000), páginas 475–486.

[Rathi et ál. 1990] A. Rathi, H. Lu y G. E. Hedrick, «Performance Comparison of Extendable Hashing and Linear Hashing Techniques», *Proc. ACM SIGSmall/PC Symposium on Small Systems* (1990), páginas 178–185.

[Reed 1983] D. Reed, «Implementing Atomic Actions on Decentralized Data», *Transactions on Computer Systems*, Volumen 1, Número 1 (1983), páginas 3–23.

[Revesz 2002] P. Revesz, *Introduction to Constraint Databases*, Springer Verlag (2002).

[Richardson et ál. 1987] J. Richardson, H. Lu y K. Mikkilineni, «Design and Evaluation of Parallel Pipelined Join Algorithms», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1987).

[Rivest 1976] R. L. Rivest, «Partial Match Retrieval Via the Method of Superimposed Codes», *SIAM Journal of Computing*, Volumen 5, Número 1 (1976), páginas 19–50.

[Robinson 1981] J. Robinson, «The k-d-B Tree: A Search Structure for Large Multidimensional Indexes», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1981), páginas 10–18.

[Roos 2002] R. M. Roos, *Java Data Objects*, Pearson Education (2002).

[Rosch 2003] W. L. Rosch, *The Winn L. Rosch Hardware Bible*, 6th edition, Que (2003).

[Rosenthal and Reiner 1984] A. Rosenthal and D. Reiner, «Extending the Algebraic Framework of Query Processing to Handle Outerjoins», *Proc. of the International Conf. on Very Large Databases* (1984), páginas 334–343.

[Ross 1990] K. A. Ross, «Modular Stratification and Magic Sets for DATALOG Programs with Negation», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990).

[Ross 1999] S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists*, Harcourt/Academic Press (1999).

[Ross and Srivastava 1997] K. A. Ross and D. Srivastava, «Fast Computation of Sparse Datacubes», *Proc. of the International Conf. on Very Large Databases* (1997), páginas 116–125.

[Ross et ál. 1996] K. Ross, D. Srivastava y S. Sudarshan, «Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Rothermel and Mohan 1989] K. Rothermel and C. Mohan, «ARIES/NT: A Recovery Method Based on Write-Ahead Logging for Nested Transactions», *Proc. of the International Conf. on Very Large Databases* (1989), páginas 337–346.

[Roy et ál. 2000] P. Roy, S. Seshadri, S. Sudarshan y S. Bhobhe, «Efficient and Extensible Algorithms for Multi-Query Optimization», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2000).

[Rusinkiewicz and Sheth 1995] M. Rusinkiewicz and A. Sheth. «Specification and Execution of Transactional Workflows», *Kim [1995]*, páginas 592–620 (1995).

[Rustin 1972] R. Rustin, *Data Base Systems*, Prentice Hall (1972).

[Rys 2001] M. Rys, «Bringing the Internet to Your Database: Using SQL Server 2000 and XML to Build Loosely-Coupled Systems», *Proc. of the International Conf. on Data Engineering* (2001), páginas 465–472.

[Rys 2003] M. Rys. «XQuery and Relational Database Systems», H. Katz, editor, *XQuery From the Experts*, páginas 353–391. Addison Wesley (2003).

[Rys 2004] M. Rys. «What's New in FOR XML in Microsoft SQL Server 2005». http://msdn.microsoft.com/en-us/library/ms345137(SQL.90).aspx (2004).

[Sagiv and Yannakakis 1981] Y. Sagiv and M. Yannakakis, «Equivalence among Relational Expressions with the Union and Difference Operators», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1981).

[Salton 1989] G. Salton, *Automatic Text Processing*, Addison Wesley (1989).

[Samet 1990] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison Wesley (1990).

[Samet 1995a] H. Samet, «General Research Issues in Multimedia Database Systems», *ACM Computing Survey*, Volumen 27, Número 4 (1995), páginas 630–632.

[Samet 1995b] H. Samet. «Spatial Data Structures», Kim [1995], páginas 361–385 (1995).

[Samet 2006] H. Samet, *Foundations of Multidimenional and Metric Data Structures*, Morgan Kaufmann (2006).

[Samet and Aref 1995] H. Samet and W. Aref. «Spatial Data Models and Query Processing», *Kim [1995]*, páginas 338–360 (1995).

[Sanders 1998] R. E. Sanders, *ODBC 3.5 Developer's Guide*, McGraw-Hill (1998).

[Sarawagi 2000] S. Sarawagi, «User-Adaptive Exploration of Multidimensional Data», *Proc. of the International Conf. on Very Large Databases* (2000), páginas 307–316.

[Sarawagi et ál. 2002] S. Sarawagi, A. Bhamidipaty, A. Kirpal, and C. Mouli, «ALIAS: An Active Learning Led Interactive Deduplication System», *Proc. of the International Conf. on Very Large Databases* (2002), páginas 1103–1106.

[Schlageter 1981] G. Schlageter, «Optimistic Methods for Concurrency Control in Distributed Database Systems», *Proc. of the International Conf. on Very Large Databases* (1981), páginas 125–130.

[Schneider 1982] H. J. Schneider, «Distributed Data Bases», *Proc. of the International Symposium on Distributed Databases* (1982).

[Selinger et ál. 1979] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie y T. G. Price, «Access Path Selection in a Relational Database System», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1979), páginas 23–34.

[Sellis 1988] T. K. Sellis, «Multiple Query Optimization», *ACM Transactions on Database Systems*, Volumen 13, Número 1 (1988), páginas 23–52.

[Sellis et ál. 1987] T. K. Sellis, N. Roussopoulos y C. Faloutsos, «TheR$^+$-Tree: A Dynamic Index for Multi-Dimensional Objects», *Proc. of the International Conf. on Very Large Databases* (1987), páginas 507–518.

[Seshadri et ál. 1996] P. Seshadri, H. Pirahesh y T. Y. C. Leung, «Complex Query Decorrelation», *Proc. of the International Conf. on Data Engineering* (1996), páginas 450–458.

[Shafer et ál. 1996] J. C. Shafer, R. Agrawal y M. Mehta, «SPRINT: A Scalable Parallel Classifier for Data Mining», *Proc. of the International Conf. on Very Large Databases* (1996), páginas 544–555.

[Shanmugasundaram et ál. 1999] J. Shanmugasundaram, G. He, K. Tufte, C. Zhang, D. DeWitt y J. Naughton, «Relational Databases for Querying XML Documents: Limitations and Opportunities», *Proc. of the International Conf. on Very Large Databases* (1999).

[Shapiro 1986] L. D. Shapiro, «Join Processing in Database Systems with Large Main Memories», *ACM Transactions on Database Systems*, Volumen 11, Número 3 (1986), páginas 239–264.

[Shasha and Bonnet 2002] D. Shasha and P. Bonnet, *Database Tuning: Principles, Experiments, and Troubleshooting Techniques*, Morgan Kaufmann (2002).

[Silberschatz 1982] A. Silberschatz, «A Multi-Version Concurrency Control Scheme With No Rollbacks», *Proc. of the ACM Symposium on Principles of Distributed Computing* (1982), páginas 216–223.

[Silberschatz and Kedem 1980] A. Silberschatz and Z. Kedem, «Consistency in Hierarchical Database Systems», *Journal of the ACM*, Volumen 27, Número 1 (1980), páginas 72–80.

[Silberschatz et ál. 1990] A. Silberschatz, M. R. Stonebraker y J. D. Ullman, «Database Systems: Achievements and Opportunities», *ACM SIGMOD Record*, Volumen 19, Número 4 (1990).

[Silberschatz et ál. 1996] A. Silberschatz, M. Stonebraker y J. Ullman, «Database Research: Achievements and Opportunities into the 21st Century», Technical Report CS-TR-96-1563, Department of Computer Science, Stanford University, Stanford (1996).

[Silberschatz et ál. 2008] A. Silberschatz, P. B. Galvin y G. Gagne, *Operating System Concepts*, 8th edition, John Wiley and Sons (2008).

[Simmen et ál. 1996] D. Simmen, E. Shekita y T. Malkemus, «Fundamental Techniques for Order Optimization», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 57–67.

[Skeen 1981] D. Skeen, «Non-blocking Commit Protocols», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1981), páginas 133–142.

[Soderland 1999] S. Soderland, «Learning Information Extraction Rules for Semistructured and Free Text», *Machine Learning*, Volumen 34, Número 1–3 (1999), páginas 233–272.

[Soo 1991] M. Soo, «Bibliography on Temporal Databases», *ACM SIGMOD Record*, Volumen 20, Número 1 (1991), páginas 14–23.

[SQL/XML 2004] SQL/XML. «ISO/IEC 9075-14:2003, Information Technology: Database languages: SQL.Part 14: XML-Related Specifications (SQL/XML)» (2004).

[Srikant and Agrawal 1996a] R. Srikant and R. Agrawal, «Mining Quantitative Association Rules in Large Relational Tables», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996).

[Srikant and Agrawal 1996b] R. Srikant and R. Agrawal, «Mining Sequential Patterns: Generalizations and Performance Improvements», *Proc. of the International Conf. on Extending Database Technology* (1996), páginas 3–17.

[Stam and Snodgrass 1988] R. Stam and R. Snodgrass, «A Bibliography on Temporal Databases», *IEEE Transactions on Knowledge and Data Engineering*, Volumen 7, Número 4 (1988), páginas 231–239.

[Stinson 2002] B. Stinson, *PostgreSQL Essential Reference*, New Riders (2002).

[Stonebraker 1986] M. Stonebraker, «Inclusion of New Types in Relational Database Systems», *Proc. of the International Conf. on Data Engineering* (1986), páginas 262–269.

[Stonebraker and Rowe 1986] M. Stonebraker and L. Rowe, «The Design of POSTGRES», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1986).

[Stonebraker et ál. 1989] M. Stonebraker, P. Aoki y M. Seltzer, «Parallelism in XPRS», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1989).

[Stonebraker et ál. 1990] M. Stonebraker, A. Jhingran, J. Goh y S. Potamianos, «On Rules, Procedure, Caching and Views in Database Systems», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990), páginas 281–290.

[Stuart et ál. 1984] D. G. Stuart, G. Buckley y A. Silberschatz, «A Centralized Deadlock Detection Algorithm», Technical report, Department of Computer Sciences, University of Texas, Austin (1984).

[Tanenbaum 2002] A. S. Tanenbaum, *Computer Networks*, 4th edition, Prentice Hall (2002).

[Tansel et ál. 1993] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev y R. Snodgrass, *Temporal Databases: Theory, Design and Implementation*, Benjamin Cummings (1993).

[Teorey et ál. 1986] T. J. Teorey, D. Yang y J. P. Fry, «A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model», *ACM Computing Survey*, Volumen 18, Número 2 (1986), páginas 197–222.

[Thalheim 2000] B. Thalheim, *Entity-Relationship Modeling: Foundations of Database Technology*, Springer Verlag (2000).

[Thomas 1996] S. A. Thomas, *IPng and the TCP/IP Protocols: Implementing the Next Generation Internet*, John Wiley and Sons (1996).

[Traiger et ál. 1982] I. L. Traiger, J. N. Gray, C. A. Galtieri y B. G. Lindsay, «Transactions and Consistency in Distributed Database Management Systems», *ACM Transactions on Database Systems*, Volumen 7, Número 3 (1982), páginas 323–342.

[Tyagi et ál. 2003] S. Tyagi, M. Vorburger, K. McCammon y H. Bobzin, *Core Java Data Objects*, prenticehall (2003).

[Umar 1997] A. Umar, *Application (Re)Engineering : Building Web-Based Applications and Dealing With Legacies*, Prentice Hall (1997).

[UniSQL 1991] *UniSQL/X Database Management System User's Manual: Release 1.2*. UniSQL, Inc. (1991).

[Verhofstad 1978] J. S. M. Verhofstad, «Recovery Techniques for Database Systems», *ACM Computing Survey*, Volumen 10, Número 2 (1978), páginas 167–195.

[Vista 1998] D. Vista, «Integration of Incremental View Maintenance into Query Optimizers», *Proc. of the International Conf. on Extending Database Technology* (1998).

[Vitter 2001] J. S. Vitter, «External Memory Algorithms and Data Structures: Dealing with Massive Data», *ACM Computing Surveys*, Volumen 33, (2001), páginas 209–271.

[Walsh et ál. 2007] N. Walsh et ál. «XQuery 1.0 and XPath 2.0 Data Model». http://www.w3.org/TR/xpath-datamodel. currently a W3C Recommendation (2007).

[Walton et ál. 1991] C. Walton, A. Dale y R. Jenevein, «A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins», *Proc. of the International Conf. on Very Large Databases* (1991).

[Weikum 1991] G. Weikum, «Principles and Realization Strategies of Multilevel Transaction Management», *ACM Transactions on Database Systems*, Volumen 16, Número 1 (1991).

[Weikum et ál. 1990] G. Weikum, C. Hasse, P. Broessler y P. Muth, «Multi-Level Recovery», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1990), páginas 109–123.

[Wilschut et ál. 1995] A. N. Wilschut, J. Flokstra y P. M. Apers, «Parallel Evaluation of Multi-Join Queues», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 115–126.

[Witten and Frank 1999] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann (1999).

[Witten et ál. 1999] I. H. Witten, A. Moffat y T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and images*, 2nd edition, Morgan Kaufmann (1999).

[Wolf 1991] J. Wolf, «An Effective Algorithm for Parallelizing Hash Joins in the Presence of Data Skew», *Proc. of the International Conf. on Data Engineering* (1991).

[Wu and Buchmann 1998] M. Wu and A. Buchmann, «Encoded Bitmap Indexing for Data Warehouses», *Proc. of the International Conf. on Data Engineering* (1998).

[Wu et ál. 2003] Y. Wu, J. M. Patel y H. V. Jagadish, «Structural Join Order Selection for XML Query Optimization», *Proc. of the International Conf. on Data Engineering* (2003).

[X/Open 1991] *X/Open Snapshot: X/Open DTP: XA Interface*. X/Open Company, Ltd. (1991).

[Yan and Larson 1995] W. P. Yan and P. A. Larson, «Eager Aggregation and Lazy Aggregation», *Proc. of the International Conf. on Very Large Databases* (1995).

[Yannakakis et ál. 1979] M. Yannakakis, C. H. Papadimitriou y H. T. Kung, «Locking Protocols: Safety and Freedom from Deadlock», *Proc. of the IEEE Symposium on the Foundations of Computer Science* (1979), páginas 286–297.

[Zaharioudakis et ál. 2000] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh y M. Urata, «Answering Complex SQL Queries using Automatic Summary Tables», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2000), páginas 105–116.

[Zeller and Gray 1990] H. Zeller and J. Gray, «An Adaptive Hash Join Algorithm for Multiuser Environments», *Proc. of the International Conf. on Very Large Databases* (1990), páginas 186–197.

[Zhang et ál. 1996] T. Zhang, R. Ramakrishnan y M. Livny, «BIRCH: An Efficient Data Clustering Method for Very Large Databases», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1996), páginas 103–114.

[Zhou and Ross 2004] J. Zhou and K. A. Ross, «Buffering Database Operations for Enhanced Instruction Cache Performance», *Proc. of the ACM SIGMOD Conf. on Management of Data* (2004), páginas 191–202.

[Zhuge et ál. 1995] Y. Zhuge, H. García-Molina, J. Hammer y J. Widom, «View maintenance in a warehousing environment», *Proc. of the ACM SIGMOD Conf. on Management of Data* (1995), páginas 316–327.

[Ziauddin et ál. 2008] M. Ziauddin, D. Das, H. Su, Y. Zhu y K. Yagoub, «Optimizer plan change management: improved stability and performance in Oracle 11g», *Proceedings of the VLDB Endowment*, Volumen 1, Número 2 (2008), páginas 1346–1355.

[Zikopoulos et ál. 2004] P. Zikopoulos, G. Baklarz, D. deRoos y R. B. Melnyk, *DB2 Version 8: The Official Guide*, IBM Press (2004).

[Zikopoulos et ál. 2007] P. Zikopoulos, G. Baklarz, L. Katsnelson y C. Eaton, *IBM DB2 9 New Features*, McGraw-Hill (2007).

[Zikopoulos et ál. 2009] P. Zikopoulos, B. Tassi, G. Baklarz y C. Eaton, *Break Free with DB2 9.7*, McGraw-Hill (2009).

[Zilio et ál. 2004] D. C. Zilio, J. Rao, S. Lightstone, G. M. Lohman, A. Storm, C. García-Arellano y S. Fadden, «DB2 Design Advisor: Integrated Automatic Physical Database Design», *Proc. of the International Conf. on Very Large Databases* (2004), páginas 1087–1097.

[Zloof 1977] M. M. Zloof, «Query-by-Example: A Data Base Language», *IBM Systems Journal*, Volumen 16, Número 4 (1977), páginas 324–343.

Índice analítico

.jar, archivos, 72
.NET Common Language Runtime (CLR)
agregados, 600-601
conceptos básicos, 599
funciones de tabla, 600-601
hospedaje de SQL, 599-600
rutinas, 600-601
SQL Server de Microsoft, 599-601
tipos, 601
Netezza, 383
.NET, 76
2PC. Véase Protocolo de compromiso de dos fases
3NF. Véase Tercera forma normal
3PC. Véase Protocolo de compromiso de tres fases
abstracción de datos, 3-4
acceso al agrupamiento asociado, 558
acceso aleatorio, 198
acceso de clave múltiple, 233-234
acciones no declarativas, 71
aceleración lineal, 365-366
ACID, propiedades. Véase atomicidad; consistencia; durabilidad; aislamiento
Active Server Pages (ASP), 179
actualización por lotes, 487-488
actualizaciones, 45
aislamiento de instantánea, 322-324
ajuste de rendimiento, 487-489, 493-494
almacenes de datos, 420-421
árboles B^+, 225-228, 229-230
asociación, 237-240
autorización, 66
bases de datos distribuidas, 389-390
complejidad, 229
control de concurrencia, 408-409
disparadores, 81-82
EXEC SQL, 77
índices, 220
lenguajes de programación persistente, 459-460
móviles, 513-514
optimización de consultas, 284-285
Oracle, 561-562
pérdidas, 322
PostgreSQL, 538, 542-544, 545-546
privilegios, 64-65
registros de registro, 337-340
SQL Server de Microsoft, 589, 592
tiempo de borrado, 225, 227-229
tiempo de inserción, 225-227, 229
vistas, 56-58
XML, 601-602
Administrador de bases de datos (DBA), 13-14, 67, 548, 578-579, 594
ADO.NET, 76, 178, 597, 599
AES (Advanced Encryption Standard), 186
agregación
.NET Common Language Runtime (CLR), 600-601
álgebra relacional y, 104-107
características avanzadas, 85-87
clasificación y, 85-87
DB2 de IBM y, 576-577
mantenimiento de vistas y, 283-284
modelo entidad-relación (E-R), 135-137
notación alternativa para, 137-140
OLAP y, 88-92
optimización de consultas y, 277
paralelismo intraoperación y, 381
PostgreSQL y, 549
procesado de consultas y, 262-263
representación de, 137
ventanas, 87
agregación, funciones
básicas, 38-39
cláusula having, 39
con agrupamiento, 38-39
fusión, 455
SQL, 38
valores Boolean, 40
valores *null*, 40
agrupación jerárquica, 428-429
agrupamientos, 366
aglomerativos, 428-429
computación en la nube, 408
DB2 de IBM, 574-575
división, 428-429
jerárquicos, 428-429
minería de datos, 422, 428-429
multidimensional, 574-575
Oracle, 558, 564
Real Application Clusters (RAC), 564
aislamiento, 2, 518
asignación de recursos, 297
atomicidad, 301-302
bloqueo, 303
control de concurrencia, 295, 297-298, 302, 541-542
definición, 293
estado inconsistente, 295
factoriales, 298
instantánea, 303-304, 322-324, 327, 338, 493, 593-594
lectura comprometida, 302, 493
lectura fantasma, 541
lectura no comprometida, 302-303
lectura no repetible, 541
lectura repetida, 302
lectura sucia, 541
marcas de tiempo, 303-304
mejora del rendimiento, 297
niveles, 302-304
Oracle, 562-563
planificación recuperable, 301
planificación sin cascada, 301-302
PostgreSQL, 541, 543
secuencialidad, 298, 302-304
tiempo de espera, 297
transacciones, 293, 297-298, 301-304
transacciones distribuidas, 391-392
utilización, 297
versionado de fila, 594
versiones múltiples, 303-304, 541-542
aislamiento de instantánea, 303-304, 541-542
secuencialidad, 323-324
sistemas de recuperación, 338-339
SQL Server de Microsoft, 594
validación, 322-323
Ajax, 176-177, 179, 408
ajuste de curva, 426
ajuste de rendimiento
actualizaciones, 487-489
ajuste de parámetros, 487, 490
carga en masa, 488-489

del esquema, 491-492
diseño físico, 492-493
elección de RAID, 490-491
hardware, 490-491
índices, 491-492
orientación de conjunto, 487-488
simulación, 494-495
transacciones concurrentes, 492-494
ubicaciones de los cuellos de botella, 489
vistas materializadas, 492
álgebra relacional, 24-25, 110-111, 195
asignación, 104
avg, 105
composición de operaciones relacionales, 98-99
count-distinct, 105
definiciones formales, 102
diferencia de conjuntos, 99
equivalencia, 271-273, 279-280
estrategia de semirreunión, 402-403
expresiones de reunión, 107
funciones agregadas, 104-107
intersección de conjuntos, 102
max, 105
min, 105
multiconjunto, 106
operación proyección, 98
operación select, 97-98
operación union, 98-99
operaciones fundamentales, 97-102
optimización de consultas, 269-274
potencia expresiva de los lenguajes, 109
procesamiento de consultas, 269-270
producto cartesiano, 99-101
proyección generalizada, 104
renombrado, 101-102
reunión externa, 104-105
reunión natural, 102-103
SQL, 98, 107
sum, 104-106
transformación de expresiones, 271-274
álgebra relacional multiconjunto, 106
algoritmo de ordenación por mezcla externo, 254
algoritmo Rijndael, 186
algoritmos de elección, 400-401
alias, 35, 160, 391, 410-411, 587
almacén de datos, 419
actualización, 421
almacenamiento orientado a columna, 421-422
componentes, 420-421
DB2 de IBM, 570, 581-582
definición, 420
desduplicación, 420-421
operación de mezcla y purga, 420-421
Oracle, 551, 553, 556-558, 560, 566
SQL Server de Microsoft, 604
tablas de hechos, 420-421
transformaciones, 420
vistas materializadas, 557-558
almacenamiento, 195
acceso a los datos, 336-337
acceso directo, 196
acceso secuencial, 196, 198
almacenes de datos, 419
archivo, 196
archivos planos, 476-477
arrays redundantes de discos independientes (RAID), 197, 200-204
atomicidad, 295-296
autorización, 9
basado en la nube, 365, 405-406
bases de datos distribuidas, 389-391
bases de datos relacionales, 477-478
cantidad de bytes, 9
cinta, 196, 205
copia de seguridad, 196, 336, 352-353, 400, 519
coste por bit, 196
datos no relacionales, 476
DB2 de IBM, 572
diccionario de datos, 210-211
discos de estado sólido, 195
discos duros, 13-14
discos magnéticos, 195, 196-200
división al nivel de bit, 201-202
durabilidad, 295-296
espejo, 201, 595-596
estable, 295, 335-336
Exadata, 565
fallos, 212-213 (véase también fallos)
flash, 182, 195, 200, 233
fragmentación, 389-391
gestión de la memoria intermedia, 9 (véase también memoria intermedia)
gestor de almacenamiento automático, 565
gestor de archivos, 9
gestor de integridad, 9
gestor de transacciones, 9 (véase también transacciones)
índices, 10 (véase también índices)
jukebox, 196
lenguajes de programación persistente, 458-459
medio físico, 195-196
memoria principal, 195-196
minería de datos, 12, 422-429
nativos, 478
no volátil, 196, 295, 335-337, 345-346
óptico, 195, 205
Oracle, 553-558, 564-565
organización de archivos, 206-210
organización del código de corrección de errores (ECC), 202-203
orientado a columna, 421-422
PostgreSQL, 545-547
procesador de consultas, 10
puntos de comprobación, 340-341, 345-346
recuperación de información, 433-443
replicación, 389-390
salida forzada, 336-337
segmentos, 553
sistemas de almacenamiento-decisión, 419-420
sistemas de copia de seguridad remota, 336, 352-353, 400, 519
sistemas de recuperación, 335-337 (véase también sistemas de recuperación)
sistemas distribuidos, 368-370
sistemas paralelos, 365-368
SQL Server de Microsoft, 589-590
tarjetas perforadas, 13
tasa de transferencia de datos, 197-198
terciario, 196, 205
tiempo de búsqueda, 196-200, 205-206, 250, 256
transparencia, 390-391
valores clob, 477
volátil, 196, 295, 335
volcado del contenido, 345
volcado, 345-346
XML, 476-479
almacenamiento de acceso directo, 196
almacenamiento de acceso secuencial, 196, 198
almacenamiento de datos y lenguaje de definición, 5
almacenamiento en cinta, 196, 205-206
almacenamiento flash, 182
árboles B^+, 232
coste, 200
NAND, 195, 200
NOR, 195, 200
unidades de disco híbridas, 200-201
velocidad de borrado, 200
almacenamiento no volátil, 196, 295, 335-337, 345-346
almacenamiento óptico, 195-196, 205
almacenamiento orientado a columna, 421-422
almacenamiento terciario, 196, 205
almacenamiento volátil, 196, 295, 335
alta disponibilidad, 352
análisis
carga en masa, 488-489
diseño de aplicaciones, 175
procesamiento de consultas, 249-250, 265, 590-591
análisis, paso de, 350
análisis de datos
almacenes de datos, 419-421
minería de datos, 422-429
recuperación de información, 433-434
sistema de soporte a la decisión, 419-421
anidamiento
ARIES, 352
bases de datos basadas en objetos, 449-450, 454-455

consultas, 279-282, 474-475, 478-480
control de concurrencia, 316, 330
DB2 de IBM, 571, 576, 580
granularidad, 316
Oracle, 552, 554, 562
transacciones, 517, 527-529, 580
transacciones de larga duración, 527
XML, 12, 465-476
ANSI (American National Standards Institute), 27, 497
any, palabra clave, 439
Apache, 174, 180, 192, 464
API (interfaz de programación de aplicaciones)
ADO.NET, 76, 498
arquitecturas de sistema, 363
C++, 499
DB2 de IBM, 570
diseño de aplicaciones, 173-174, 178
DOM, 481
Java, 71-74, 94, 173-174, 480, 487
LDAP, 481
mapas personalizados, 506-507
normas para, 497-499
ODBC, 74-76, 498
PostgreSQL, 536
SAX, 481
SQL Server de Microsoft, 587, 595-597, 599-600, 604-605
XML, 466, 476
aplicaciones multisistema, 519
Apple Macintosh OS X, 535
apply, operador, 588
árbol de información de directorio (DIT), 410-413
árbol equilibrado, 222
árboles, 514
B, 232-233, 244, 491, 504, 507-508, 510, 514, 540, 546-547, 552, 554-556, 558, 574 (véase también árboles B)
B^+, 6-16, 590 (véase también árboles B^+)
clasificador de árboles de decisión, 423-425
directorio distribuido, 412-413
división cuadrática, 509-510
Generalized Search Tree (GiST), 546-547
granularidad múltiple, 316-318
información de directorio (DIT), 410-413
k-d, 508
optimización de consultas, 382-383 (véase también optimización de consultas)
Oracle, 554-555, 567
planificación, 382-383
PostgreSQL, 546-547
quadtrees, 506, 508-509
R, 509-510
sobreajuste, 425
soporte a datos espaciales, 505-510
XML, 472, 477
árboles B
datos espaciales, 504, 508, 510, 514
DB2 de IBM, 574
desarrollo de aplicaciones, 492
índices, 232, 244, 492
Oracle, 552, 554-556, 558
PostgreSQL, 540, 546-547
árboles B^+, 243, 509
actualización, 225-229
árboles equilibrados, 222
blob, 62, 74, 208, 230, 478, 571, 602
bloque de reunión de bucle anidado, 255-256
carga masiva de índices, 231-232
claves de búsqueda no únicas, 228-229
consultas en, 224-226, 249, 252
división en el nivel de bloque, 201-202
escrituras ciegas, 320
estructura, 242-243
existencia, 242-243
exploración, 548
extensiones, 230-233
indexado de cadenas de textos, 231
índices de mapas de bits, 243
índices secundarios, 231
memoria flash, 233
nodos internos, 222
operaciones booleanas, 37, 40-42, 72, 79, 412, 600. Véase también operaciones específicas
operaciones eficientes, 243
Oracle, 555-556
organización de archivos, 230-231
punteros, 222
registros secuenciales, 241-242
rendimiento, 222
reubicación de registros, 231
sobre claves múltiples, 233
tiempo de borrado, 225, 227-228, 229-230
tiempo de inserción, 225-227, 229-230
árboles k-d, 508
árboles R, 509-510
archivos secuenciales, 209
ARIES, 545
acciones superiores anidadas, 352
bloqueo de grano fino, 352
estructura de datos, 349-350
números de secuencia de registro (LSN), 349-351
optimización, 352
paso de análisis, 350
paso de deshacer, 351-352
paso de rehacer, 351
puntos de comprobación difusos, 349
recuperación, 349-350, 352
registros del registro de compensación (CLR), 349, 351
rehacer fisiológico, 349
tabla de páginas sucias, 349-351
vuelta atrás de transacciones, 351
Arpanet, 371
arquitectura centralizada, 361-362
arquitectura de disco compartido, 368-370
arquitectura de dos capas, 11
arquitectura de memoria compartida, 367-368
arquitectura de no compartición, 367-369
arquitectura del sistema. Véase arquitectura
arquitectura en tres capas, 11
arquitectura hipercubo, 366
arquitectura jerárquica, 367-368
arquitectura(s), 361
almacén de datos, 420-421
bancos de hebras, 596
bases de datos paralelas, 375-385
centralizada, 361-362
cliente-servidor, 362-363
disco compartido, 367-368, 370
dos capas, 11
en la nube, 365
hipercubo, 367
jerárquica, 367-368
lógica de negocio, 11, 551, 581-582, 587, 589, 599, 604-605
malla, 366-367
memoria compartida, 367-368
monitor de procesos, 363
monitor TP, 517-519
protocolo de compromiso de dos fases (2PC), 370
red de almacenamiento (SAN), 370
red de área extensa (WAN), 370-372
servidor compartido, 564
servidor de datos, 363-365
servidor de transacciones, 363-364
sin compartición, 367-369, 377
sistema distribuido, 368-370
sistema para un usuario, 362
sistema servidor, 363-365
sistemas paralelos, 365-368
tipos de redes, 370-372
tres capas, 11
array, tipos, 453-455
as, cláusula, 34-35
asc, expresión, 35-36
aserciones, 5, 61
asignación, operación, 79, 97, 104, 498
asociación muchos a muchos, 122, 124-125
asociación muchos a uno, 122, 124
asociaciones, 8, 21
conjuntos de entidades, 131 (véase también conjuntos de entidades)
conjuntos de relaciones, 119-120, 139-140, 142
propiedad asociativa, 271-272
reglas, 427-428

ASP.NET, 174
Aster Data, 383
ataques de diccionario, 186
ataques de hombre en medio, 183, 523
atomicidad, 2-3, 10-11, 46
aislamiento, 301-302
definida, 293
estructura de almacenamiento, 295-296
flujos de trabajo, 520-521
planificación sin cascada, 301-302
planificaciones recuperables, 301
protocolos de compromiso, 392-395
registros del registro, 337-339, 341
sistemas de recuperación, 337, 341
transacciones distribuidas, 391-392
atributos
acceso de clave múltiple, 233-234
anidamiento, 454-456
cierre de conjunto de atributos, 154
clasificadores, 424
clave de búsqueda, 217
complejos, 125, 128-129
compuestos, 120
conjunto de valores, 120
conjunto de entidades, 128-129, 131
derivados, 121
descomposición, 149-153, 157-161
diagramas entidad-relación, 125
dominio, 120
dominios atómicos, 149
elementos de diseño, 131
Lenguaje unificado de modelado (UML), 139-140
modelo entidad-relación (E-R), 119-121
multivalorados, 120-121, 149
nombrado, 162-163
particiones, 424
simples, 120, 128
sin anidamiento, 454-455
tipos de XML, 469-472
ubicación de, 132-133
valor único, 120-121
valores continuos, 424
valores nulos, 121
atributos de un único valor, 120-121
atributos determinados funcionalmente, 154
atributos multivalorados, 121, 149
auditorías, pruebas, 185
autenticación
certificados digitales, 187-188
cifrado, 187-188
firma digital, 187-188
seguridad, 183-184 (véase también seguridad)
sistema de desafío-respuesta, 187
sistema de firma única, 183-184
tarjetas inteligentes, 187-188
autonomía, 403
autorización, 5, 9, 27
actualización, 66
administradores, 64
cláusula check, 66
concesión/revocación de privilegios, 13, 64-65, 67
diseño de bases de datos, 141
esquema, 66
falta de grano fino, 184
información de usuario final, 184
Lenguaje de marcado de aserciones de seguridad (SAML), 183
nivel de aplicación, 184
roles, 65-66
seguridad sql, 66
transferencia de privilegios, 66-67
vistas, 65-66
avg, expresión, 90
álgebra relacional, 105
funciones de agregación, 38-39
procesado de consultas, 262-263
bancos de hebras, 596
bancos de memoria intermedia, 572-573, 581
BASE, propiedades, 401
bases de datos (véase también bases de datos basadas en objetos; bases de datos especiales; bases de datos relacionales)
abstracción, 3-5
almacenamiento, 9-10, 195 (véase también almacenamiento)
arquitectura, 13-14, 361 (véase también arquitecturas)
bloqueos, 309-316 (véase también bloqueos)
control de concurrencia, 309-330 (véase también control de concurrencia)
distribuidas, 389-413, 565 (véase también bases de datos distribuidas)
historia, 13-14
indexación, 217-245 (véase también índices)
memoria intermedia, 343-344 (véase también memoria intermedia)
memoria principal, 524-525
modificación, 43-45, 339
móviles, 511-514
multimedia, 510-511
normalización, 8-9
paralelas, 375-385 (véase también bases de datos paralelas)
personal, 511-514
recuperación de la información, 433-443
salida forzada, 336-337
sistema de proceso de archivos, 2-3
sistemas de recuperación, 295-296, 335-354 (véase también sistemas de recuperación)
volcado, 345-346
base de datos, ejemplar, 20
base de datos, grafo, 313-314
bases de datos basadas en objetos, 462
anidamiento, 449-450, 454-456
correspondencia, 453
enfoque orientado a objetos vs. objeto relacional, 461-462
herencia, 451-453
implementación de características, 456-457
lenguajes de programación persistente, 457-462
modelo de datos relacional, 449
no anidados, 454-455
tipos array, 453-455
tipos de datos complejos, 449-450
tipos de referencia, 455-456
tipos estructurados, 451-452
tipos multiconjuntos, 453-455
volúmenes colección, 454
Object Database Management Group (ODMG), 499
bases de datos distribuidas, 412-413
almacenamiento, 389-391
basadas en la nube, 405-409
bloqueos, 395-398
control de concurrencia, 393-394, 396-398
disponibilidad, 398-401
división, 393
fallos, 391-393
fragmentación, 389-390
heterogéneas, 389-390, 403-405
homogéneas, 389-390
manejo de bloqueos, 397-398
marcas de tiempo, 396-397
mensajes de persistencia, 394-395
procesamiento de consultas, 402-404
protocolos de compromiso, 392-395
recuperación, 393-394
replicación, 389-390, 397
reunión, 402-403
sistemas de directorio, 409-413
transparencia, 390-391
vista unificada de datos, 403-404
bases de datos especiales, 449
bases de datos basadas en objetos, 449-462
XML, 465-481
bases de datos multimedia, 510-511
bases de datos orientadas a objetos, 177
bases de datos paralelas
aumento de su uso, 375
consultas de soporte a la decisión, 375
coste, 375
diseño de sistema, 382-383
encauzamiento, 382-383
éxito, 375
memoria caché, 384
multihebra, 384

optimización de consultas, 382-383
Oracle, 560-561
paralelismo de E/S, 375-377
paralelismo interconsulta, 377-378
paralelismo interoperación, 382
paralelismo intraconsulta, 377-378
paralelismo intraoperación, 378-381
procesadores masivamente paralelos (MPP), 569
procesadores multinúcleo, 383-384
tasas de fallo, 383
técnicas de división, 375-376
velocidad en bruto, 383-384
bases de datos relacionales
acceso desde programas de aplicación, 7
almacenamiento, 477-478
lenguaje de definición de datos, 6
lenguaje de manipulación de datos (DML), 6
tablas, 6
Bayes, teorema, 425
BCNF. Véase forma normal de Boyce-Codd
begin atomic...end, 58, 79, 81-82
biblioteca de etiquetas, 175
biblioteca YUI, 176
big-bang, enfoque, 497
Bigtable, 405-408
bits de paridad, 201-203
bloqueo en modo compartido, 309
bloqueo en modo exclusivo, 309
bloqueos
bases de datos distribuidas, 395-398
caché, 364
ciclos falsos, 398
compartidos, 309, 396
concesión, 311-312
consistencia, 310-311
control de concurrencia, 309-314
DB2 de IBM, 579-581
de grano fino, 352
dinámicos, 594
esquemas multiversión, 322
exclusivos, 303, 309, 312-314, 316, 322, 325-327, 328-331, 338-339, 343-344, 377, 395-396
fase de crecimiento, 311-312
función de compatibilidad, 309
grafo de espera, 315-316, 398
granularidad adaptativa, 364
granularidad múltiple, 316-318
implementación, 312-313
inanición, 316
interbloqueo, 310-311, 314-316, 395-398, 579-581, 594-596
manejo, 314-317
marcas de tiempo, 318-319
modos de intención, 317
niveles alto/bajo, 346
operación de deshacer lógica, 345-348
operaciones de petición, 309-313, 315-317, 330
PostgreSQL, 543-545
prevención, 315-316
selección de víctima, 316
servidores de transacciones, 363-364
sistemas de recuperación, 345-348
SQL Server de Microsoft, 593-596
transacciones de larga duración, 526
vuelta atrás, 316
borrado, 29, 44-45, 72
asociación, 234, 236-237, 241
control de concurrencia, 324-326
disparadores, 81
EXEC SQL, 77
PostgreSQL, 538
privilegios, 64-65
restricciones de integridad, 60
transacciones, 294, 304
vistas, 56-57
bucle repeat, 84, 154-155, 224
bucle while, 75, 77, 79
buscadores Web, 440-441
búsqueda, 278, 514
actualización perdida, 322
aprendizaje de máquina, 12
bases de datos distribuidas, 407-409
control de concurrencia, 326, 328-329
descomposición con pérdida, 156
descomposición de reunión con pérdida, 156
difusa, 420, 605
índices, 220, 222-229, 232-236, 238, 241-242
PostgreSQL, 546
procesamiento de consultas, 252, 256
SQL Server de Microsoft, 591, 593, 605
búsqueda lineal, 251
BYNET, 379
C, 3, 6
diseño de aplicaciones, 174-175, 179-180
Lenguaje unificado de modelado (UML), 139
ODBC, 75-76
SQL avanzado, 71, 76-77, 80
SQL Server de Microsoft, 587, 599
C++, 3, 6
bases de datos basadas en objetos, 449
Lenguaje unificado de modelado (UML), 139
norma, 499
sistemas persistentes, 458-460
SQL avanzado, 76-77
SQL Server de Microsoft, 599
cabecera de archivo, 206
caché, 195
arquitectura de memoria compartida, 368
bloqueos, 364
coherencia, 364
diseño de aplicaciones, 181
multiproceso, 383-384
Oracle, 563
planes de consulta, 280, 364
servidores de datos, 364
cadenas de indexación, 231
cajones de desbordamiento, 235-236
cálculo relacional de dominios, 110
consulta de ejemplo, 109-110
definición formal, 109
particularización, 89
potencia expresiva de los lenguajes, 110
reunión por asociación de doble encauzamiento, 264-265
seguridad de las expresiones, 110
cálculo relacional de tuplas, 107, 110
consultas de ejemplo, 107-108
definición formal, 108
potencia expresiva de los lenguajes, 109
seguridad de las expresiones, 109
cambio de contexto, 517
CAP, teorema, 401
capa de acceso a los datos, 176-178
capa de lógica de negocio, 19, 176
capa de presentación, 176
capa de traducción flash, 200
cardinalidad de asociación, 121-122, 124-125
carga en masa, 231-232, 488-489
cartesiano, producto, 31
álgebra relacional, 24, 97, 99-104
consultas, 265, 271, 274, 276-277, 281, 286
equivalencia, 271
expresiones de reunión, 102-103
SQL, 31-35, 55, 92
cascada, 59, 67, 82-83
case, construcción, 45
CASE, herramientas, 570
cast, 61-63
catálogos
bases de datos distribuidas, 391, 399
DB2 de IBM, 570, 581
desarrollo de aplicaciones, 498
e-catálogos, 522-523
índices, 217 (véase también índices)
optimización de consultas, 274-276, 547
PostgreSQL, 548-549
procesamiento de transacciones, 528
recuperación de información, 433, 442
sistema, 210, 213, 376, 538
SQL Server de Microsoft, 590-591, 597, 600, 605
SQL, 64, 74, 76
XML, 480
catálogos del sistema, 210-211, 538-539
categorías, 442-443
centralizada, arquitectura, 361-362
centralizado, detección de bloqueo, 398
certificados digitales, 187-188
check, cláusula, 59-60
aserción, 61

autorización, 66
conservación de dependencia, 151-152
restricciones de datos, 140
tipos definidos por el usuario, 63
check, restricciones, 60, 66, 140, 143, 151, 293, 538
ciclos falsos, 398
cierre transitivo, 84-85
cifrado
Advanced Encryption Standard (AES), 186
algoritmo Rijndael, 186
aplicaciones, 185-188
ataques de diccionario, 187
autenticación, 187-188
certificados digitales, 187-188
de clave asimétrica, 186
de clave pública, 186-187
firma digital, 187
no repudio, 187
números primos, 186
Oracle, 554-555
sistema de desafío respuesta, 187
soporte de bases de datos, 186-187
técnicas, 186-187
cifrado de clave asimétrica, 186
cifrado de clave pública, 186-187
clasificación, 85-87
clasificación jerárquica, 442-443
clasificación por popularidad, 435-437
clasificación por prestigio, 435-437, 440-441
clasificación, jerarquía, 442-443
clasificadores
árboles de decisión, 423-425
bayesianos, 425-426, 567, 605
ejemplares de entrenamiento, 423
funciones del núcleo, 425-426
ganancia de información, 424
medida de Gini, 424
medida de la entropía, 424
mejores divisiones, 424-425
particiones, 423-424
predicción, 422-427
pureza, 424
red neuronal, 425
regresión, 426-427
validación, 426-427
clasificadores bayesianos, 425-426
clasificadores de árbol de decisión, 423-425
cláusula every, 40
cláusula except, 37-38, 41, 84
cláusula exists, 41
cláusula for each row, 81-82
cláusula for each statement, 31, 81
cláusula if, 82
cláusula pivot, 91, 93, 588
cláusula recursiva, 85
cláusula set, 45
cláusula when, 81, 82
cláusula where, 140
consultas básicas de SQL, 29-34
en relaciones múltiples, 30-33
en una única relación, 29-31
entre, 36
funciones agregadas, 38-40
no entre, 36
operación rename, 34-35
operaciones de cadena, 35-36
operaciones de conjuntos, 36-38
optimización de consultas, 280-282
reunión natural, 33-34
seguridad, 184
transacciones, 303-305
valores *null*, 37-38
cláusula with, 43, 85
clave candidata, 21-23
clave externa, 22, 28-29, 59-60
clave(s), 21-23
acceso múltiple, 233-234
almacenamiento, 208-209
asociación, 234-239, 241
cifrado, 186-189
dependencias funcionales, 150-151
descomposición, 159-160
igualdad, 251
indexado, 217-234, 242, 244
modelo entidad-relación (E-R), 122-123
no único, 228-229
restricciones, 122-123
tarjetas inteligentes, 187-188
USB, 195
claves de búsqueda
almacenamiento, 208-209
asociación, 234-239, 241
indexado, 217-234, 241, 244
no únicas, 228-229
claves primarias, 21-22, 28-29
dependencias funcionales, 150-151
descomposición, 159-160
modelo entidad-relación (E-R), 122-123
restricciones de integridad, 59-60
cliente-servidor, sistemas, 10, 15, 90
arquitectura del sistema, 361-362, 365, 370, 372
diseño de aplicaciones, 169-170, 488, 498-499
procesamiento de transacciones, 517-519
sistemas de recuperación, 352
SQL Server de Microsoft, 587
clob, 62, 74, 208, 231, 477-478, 570-571
código de byte, 175
código de corrección de errores (ECC), organización, 202-203
collect, función, 454
combinatoria, 298
comercio electrónico, 522-524
Common Language Runtime (CLR), 80
compatibility, function, 309
compensación, registros del registro (CLR), 349-351
compresión, 510-511
almacenes de datos, 422
DB2 de IBM, 570
desarrollo de aplicaciones, 492-493
Oracle, 554-556
prefijo, 231, 555
SQL Server de Microsoft, 590
compresión de carga de trabajo, 492-493
computación en la nube, 365
almacenamiento, 405-406
bases de datos tradicionales, 408
desafíos, 408-409
particiones, 407
recuperación, 407
replicación, 408
representación de datos, 405-406
transacciones, 408-409
comunicación inalámbrica, 511-514
con tiempo de zona, 61
concentrador, 437
concesión, 65-67
concesión del rol actual, 67
conectiva or, 30
confianza, 422, 427
conjunción, 252-253, 275
conjunto de entidades disjuntas, 135
conjunto de resultados no actualizables, 74
conjunto nulo, 60
conjuntos de entidades
atributos, 119, 128-129, 131
atributos simples, 128
conjunto de relaciones, 119-120, 131-132
débiles, 125-126, 128-129
definición, 118-119
definición por condiciones, 135
definición por usuario, 135
disjuntos, 135
elementos de diseño, 131-132
eliminación de redundancia, 123-124
extensión, 118
fuertes, 128-129
generalización, 134-137
identificación de relaciones, 126
Lenguaje unificado de modelado (UML), 139-140
notaciones alternativas, 137-140
propiedades, 118-119
relaciones superclases-subclases, 133-134
rol, 119-120
subclases, 134
superclases, 134
superposición, 134-135
conjuntos de entidades definidas por condiciones, 134-135

conjuntos de entidades definidas por el usuario, 134-135
conjuntos de relaciones
atributos descriptivos, 120
binario vs. *n*-ario, 132
combinación de esquema, 130-131
conjuntos entidad, 131-132
diagramas entidad-relación, 125-126
dominios atómicos, 149
elementos de diseño, 131-133
Lenguaje unificado de modelado (UML), 139-140
modelo entidad-relación (E-R), 119-120, 129-130, 133-134
no binarios, 125-126
nombrado, 162-163
notación alternativa, 137-140
recursivos, 119
redundancia, 130
representación, 129-131
superclase-subclase, 133-134
ubicación de atributos, 132-133
conjuntos de relaciones recursivas, 119
conmutativa, propiedad, 271-272
conservación de dependencias, 151-152, 156-157
consistencia, 5, 10, 46
bloqueos, 310-311
control de concurrencia, 323, 326-328, 330
disponibilidad, 401
estabilidad del cursor, 327
grado dos, 326-327
interacciones de usuario, 327-328
niveles ligeros, 326-328
operaciones lógicas, 346
redes móviles, 513-514
replicación, 397
requisitos, 294
teorema CAP, 401
transacciones, 293-295, 297-298, 302, 305
transacciones distribuidas, 391-392
consistencia de grado dos, 326-327
construcción cube, 91-93
construcción in, 41
consulta de vecino más cercano, 507-508
consulta dependiente de ubicación, 512
consultas, 5
acceso de clave múltiple, 233-234
ADO.NET, 76
árboles B^+, 223-225
asociación, 217, 237-240 (véase también funciones de asociación)
bases de datos basadas en objetos, 449-462
bases de datos distribuidas, 389-413 (véase también bases de datos distribuidas)
bases de datos paralelas, 375-385
borrado, 43-44
caché, 181
datos espaciales, 507-508
dependiente de ubicación, 512
disponibilidad, 389-390
diversidad de resultado, 441
estructura básica de SQL, 29-33
índices, 217 (véase también índices)
inserción, 44-45
JDBC, 71-75
lenguaje de definición de datos (DDL), 9-10
lenguaje de manipulación de datos (DML), 9-10
lenguaje de programación persistente, 457-461
máquina universal de Turing, 6-7
metadatos, 74
objeto ResultSet, 71-72, 74-75, 177, 179-180, 224
obtención de resultados, 72-73
ODBC, 74-76
OLAP, 88-92
on single relation, 29-31
operaciones de cadena, 35-36
operaciones de conjunto, 36-38, 40-42
Oracle, 557-558
PageRank, 436-437
producto cartesiano, 24, 31, 33-35, 55, 92, 97, 99-102, 104, 265, 271, 274, 276, 281, 286
recuperación de información, 433-444
recursivas, 83-86
requisitos de usuario, 140-141
reunión natural, 33-34, 39, 51-55 (véase también reunión)
seguridad, 181-188
servidor de transacciones, 364
servlets, 173-176
solo lectura, 378
soporte a la decisión, 375
SQL intermedio, 51-67
subconsultas anidadas, 40-43
subconsultas correlacionadas, 41
subconsultas escalares, 43
tipos de datos complejos, 449-451
valores *null*, 37-38
vecino más cercano, 507-508
vistas, 55-58
XML, 472-476
consultas de intervalos, 376
consultas de solo lectura, 378
consultas de soporte a la decisión, 375
consultas recursivas, 83
cierre transitivo, 84-85
iteración, 84-85
SQL, 85-86
contador lógico, 318
contención de lectura-escritura, 492-493
contención escritura-escritura, 493
contraseña. Véase también seguridad
almacenamiento, 210-211
ataque de diccionario, 187
ataque de hombre en medio, 183
bases de datos distribuidas, 410
de un solo uso, 183
debilidad, 182
diseño de aplicaciones, 169, 172, 174, 177, 183-184, 187
sistemas de firma de una vez, 183-184
SQL, 64, 72, 75-76
contratos de extensibilidad, 600-601
control de acceso al medio (MAC), 537
control de concurrencia, 295, 297, 298, 330 (véase también control de concurrencia multiversión)
actualizaciones, 408-409
aislamiento de instantánea, 322-324, 338-339
anomalías de acceso, 3
bases de datos distribuidas, 393-394, 395-398
bloqueos, 309-320, 395-396
buscadores Web, 440-441
ciclos falsos, 398
comandos DML, 541-542
consistencia, 323, 326-328, 330
DB2 de IBM, 572-573, 579-580
deshacer lógico, 348-349
escritura ciega, 320
esquemas multiversión, 321-322, 541-545
estructura de índices, 327-328
granularidad múltiple, 316-318
interacciones de usuario, 327-328
lectura de predicados, 324-326
manejo de bloqueos, 314-316
operación de inserción, 324-326
operaciones de borrado, 324-326
Oracle, 561-563
PostgreSQL, 541-545
protocolo basado en marcas de tiempo, 318-319
secuenciabilidad, 302, 309, 311-314, 317-321, 323-324, 326-328, 329
sistemas de recuperación, 338-339
SQL Server de Microsoft, 593-596
transacciones de larga duración, 526-527
validación, 320, 338-339
vuelta atrás, 311-312, 314-316, 319, 321-322, 330
control de concurrencia multiversión (MVCC)
bloqueos, 544
comandos DDL, 544-545
comandos DML, 541-542
esquema, 321-322
implementación, 541-544
implicaciones de uso, 543-544

índices, 545
MySQL, 14, 35, 49, 72, 535, 550
niveles de aislamiento, 541-542
recuperación, 544-545
control de concurrencia optimista sin validación de lectura, 327
control de transacciones, 27
controlador de disco, 197
cookies, 172-174, 182-183
coordinador de transacciones, 391-393, 400-401
copia de seguridad, 82-83
arquitecturas de sistemas, 361
bases de datos distribuidas, 395, 412
bases de datos paralelas, 383
DB2 de IBM, 579-580
diseño de aplicaciones, 187
Oracle, 554, 556, 562-563, 565-566
sistemas de recuperación, 337-342 (véase también sistemas de recuperación)
sistemas remotos, 352-353, 400, 519
SQL Server de Microsoft, 585, 587, 595, 603
transacciones, 295, 528
copia de seguridad, coordinador, 400
copia principal, 396
CORBA (Common Object Request Broker Architecture), 499
correlación de variables, 280-282
correlaciones, 428
correspondencia de objetos relacional, 177, 449, 461
coste de consulta
elección, 278-282
encauzamiento, 263-265
expresión, 262-265
materialización, 262-263
motor de evaluación de consultas, 10
operaciones de conjunto, 261
optimización, 269-286, 274-279
planes de evaluación de consultas, 249-250
procesamiento, 250-252, 254, 257-258, 260
SQL Server de Microsoft, 591-592
tiempo de respuesta, 251
vista, 271
costes de arranque, 366
creación de aserción, 61
creación de dominio, 63
creación de esquema, 64
creación de función, 78-79, 84
creación de índice, 243-244, 547-548
creación de índice único, 244
creación de instantánea, 397-398
creación de procedimiento, 78-80
creación de rol, 65-66
creación de tabla, 28-29, 62, 72
bases de datos basadas en objetos, 451, 455-456
con datos, 63-64
extensiones, 63-64
restricciones de integridad, 58
creación de tabla...como, 64
creación de tipo, 62-63
creación de tipo distinto, 63, 569-571
creación de vista, 55-57, 64, 66, 538
creación de vista recursiva, 85
creación o reemplazo, 78
CRUD, interfaces Web, 180
Crystal Reports, 180-181
cuarta forma normal, 160-162
cube by, 581-582
cubrimiento canónico, 154-156
cuellos de botella
arquitectura del sistema, 368, 376, 383-384, 390, 395-396
diseño de aplicaciones, 181, 487, 489-490, 492
estructura de archivo, 213
transacciones, 524-525, 528
cuenta, 38-39, 41, 262-263
data cube, 88, 91-93
Data Encryption Standard (DES), 186
Database Task Group, 498
DataGrid, control, 179
DATAllegro, 383
Datalog, 19
datetime, 89
datos espaciales
consultas, 507-508
datos de diseño asistido por computadora, 503, 505-506
datos de vectores, 506
datos geográficos, 503, 505
indexado, 507-510
información topográfica, 507
representación de información geométrica, 504-505
triangulación, 505
datos geográficos, 503
aplicaciones, 506
consulta espacial, 507-508
datos de vector, 506
datos ráster, 506
representación, 505-508
sistemas de información, 505
datos multidimensionales, 88
datos multimedia, 503
datos ráster, 506
datos temporales, 503
bases de datos relacionales, 163-165
intervalos, 504-505
lenguajes de consulta, 504
marcas de tiempo, 504-505
tiempo de las transacciones, 503
tiempo en las bases de datos, 503-505
DB2 de IBM, 14, 43, 63, 72, 77, 80, 82, 95, 535
agrupamiento multidimensional, 574-575
aislamiento, 580
almacenamiento, 572-573
arquitectura del sistema, 581
bancos de memoria intermedia, 572-573
bloqueos, 580-581
características, 578-579
características de inteligencia de negocio, 581-582
Centro de control, 570, 579
conjunto de operaciones, 576
control de concurrencia, 572-573, 580
Data Warehouse Edition, 581
datos externos, 581
desarrollo de, 569
distribución, 581
funciones definidas por el usuario, 570-571
herramientas administrativas, 579
herramientas de diseño de bases de datos, 570
indexado, 571-574
objetos grandes, 571-572
procesadores masivamente paralelos (MPP), 569
procesamiento de consultas, 277, 280, 284, 575-579
recuperación, 572-573
replicación, 581
restricciones, 571
reunión, 576
servicios Web, 571-572
Servidor de base de datos universal, 569-570
soporte de tipos de datos, 570-571
System R and, 569
utilidades, 579-580
variaciones de SQL, 570-572
vistas materializadas, 577-578
vuelta atrás, 580
XML, 570
DB-Lib, 597
declaración de identidad, 493
declare, 78-80
decodificación, 92
definición de esquema, 13
definición de tipo de documento (DTD), 469-470
definición de vistas, 27
dependencias funcionales, 8, 58
algoritmo BCNF, 157-158
cierre de atributos, 154
cierre de un conjunto, 153-154
claves, 150-151
conservación de dependencias, 151-152, 156-157
de reunión, 161
descomposición sin pérdidas, 159

forma normal de Boyce-Codd, 151-152
formas normales superiores, 152-153
multivaloradas, 160-161
recubrimiento canónico, 154-156
regla reflexiva, 153
regla transitiva, 153
regla de aumento, 153
regla de descomposición, 153
regla de unión, 153
regla pseudotransitiva, 153
teoría, 153-157
tercera forma normal, 152-153, 158-159
dependencias multivaloradas, 160-161
desafío-respuesta, sistema, 187
desarrollo de aplicaciones (véase también desarrollo rápido de aplicaciones)
actualizaciones, 487-489
ajuste de rendimiento, 487-494
normalización, 497-499
orientación a conjuntos, 487-488
prueba de aplicaciones, 496-497
pruebas de rendimiento, 494-496
desarrollo rápido de aplicaciones (RAD)
biblioteca de funciones, 179
entorno de aplicaciones Web, 179-180
generadores de informes, 180
herramientas de construcción de interfaces de usuario, 179-180
desbordamiento de tabla asociativa, 259
desc, 35-36
descomposición
algoritmos, 157-159
claves, 150-151
con pérdidas, 155-156
conservación de dependencias, 151-152
cuarta forma normal, 161
DEC Rdb, 14
dependencias funcionales, 149-153, 160-161
dependencias multivaloradas, 160-161
diseño de base de datos relacional, 149-153, 157-161
forma normal de Boyce-Codd, 151-152, 157-159
formas normales superiores, 152-153
reglas de descomposición, 153
reunión sin pérdida, 156
sin pérdida, 156
tercera forma normal, 152-153, 158-159
desduplicacion, 420-421
deshabilitar disparador, 82
deshacer
control de concurrencia, 348-349
operaciones lógicas, 346-348
Oracle, 553
sistemas de recuperación, 338-342
vuelta atrás de transacciones, 346-348
desnormalización, 163-164
desviación, 95, 427-428
detección de bloqueo centralizado, 398
diagrama entidad-relación (E-R), 8
asociación de cardinalidad, 124
atributos complejos, 124-125
conjunto de entidades, 126
conjuntos de entidades débiles, 126
conjuntos de relaciones, 125-126
conjuntos de relaciones no binarias, 125-126
ejemplo de la universidad, 127-128
estructura básica, 124
generalización, 134
identificación de relaciones, 126
notaciones alternativas, 137-140
roles, 125
diagramas de esquema, 23
diccionario de datos, 5, 10, 210-211
diferencia de conjuntos, 24, 99, 271
difusión de datos, 512-513
directorios, 442-443
disco blando, 195
disco de registro, 199-200
discos de escribir una vez, leer muchas veces (WORM), 196
discos de estado sólido, 195
discos duros, 13-14
discos duros híbridos, 200-201
discos magnéticos, 195
arrays redundantes de discos independientes (RAID), 201-204
bloqueos, 198-200
cabezas de lectura/escritura, 196-197
características físicas, 196-197
clasificación de fallos, 335
controlador de disco, 197
densidad de grabación, 196-197
disco de registro, 199-200
fallos, 197
híbridos, 200-201
lectura adelantada, 198
medidas de rendimiento, 198
memoria intermedia, 198-199
optimización del acceso a bloques de disco, 198-200
planificación, 198-199
sectores, 196-197
sistemas paralelos, 367
suma de comprobación, 197
tamaños, 196
tasa de transferencia, 198
tiempo de búsqueda, 198
tiempo medio entre fallos, 198
diseño asistido por computadora (CAD), 141, 503, 505-506
diseño conceptual, 7-8, 117
diseño de aplicaciones, 188
arquitectura cliente-servidor, 169-170
arquitectura de tres capas, 144
arquitecturas de aplicación y, 176-179
autenticación y, 183-184
capa de acceso a datos, 176-178
capa de lógica de negocio, 176-177
cifrado, 185-188
cookies, 172-174
desarrollo rápido de aplicaciones, 179-180
interfaces de usuario, 169-170
interfaz de pasarela común (CGI), 171-172
Java Server Pages (JSP), 170,173-174
Lenguaje de marcas de hipertexto (HTML), 171
Localizadores uniformes de recursos (URL), 170-171
monitores TP, 519
operación sin conexión, 178-179
Protocolo de transferencia de hipertexto (HTTP), 170-173, 178, 182-183, 188
rendimiento, 181-182
seguridad, 181-189
servlets, 173-176
World Wide Web, 170-172
diseño de arriba abajo, 134
diseño de bases de datos, 117, 141
adaptación para las arquitecturas modernas, 384
ajuste automático, 492-493
alternativas, 118-119, 137-140
aplicación, 169-189
arquitectura cliente-servidor, 15, 90, 169-170, 352, 357, 361-362, 365-366, 371
cifrado, 185-188
complejidad, 117
DB2 de IBM, 570
de abajo arriba, 134
de arriba abajo, 134
descripción del proceso, 117-118
entidad-relación (E-R)
especificación de requisitos funcionales, 7
fase de diseño conceptual, 7-8, 117
fase de diseño lógico, 7, 117-118
fase del diseño físico, 7, 117
fases de, 117-118
flujo de trabajo, 141
incompletitud, 118
memoria intermedia, 211-213
modelo, 8, 110-111
necesidades de usuario, 117
normalización, 8-9
Oracle, 551
proceso dirigido por el diseño, 117-118
redundancia, 118
relacionales, 147-165 (véase también diseño de bases de datos relacionales)
requisitos de autorización, 141
requisitos de usuario, 140-141
restricciones de datos, 140-141

sistemas paralelos, 382-383
SQL Server de Microsoft, 585-587
universidades, 7-8
diseño de base de datos relacionales, 165
características de un buen, 147-149
cuarta forma normal, 160-161
dependencias funcionales, 149-157
dependencias multivaloradas, 160-161
descomposición, 149-153, 157-161
dominios atómicos, 149
esquemas grandes, 147-148
esquemas pequeños, 148-149
modelado de datos temporales, 163-164
nombres de atributos, 162-163
nombres de relaciones, 162-163
normalización, 162
primera forma normal, 149
proceso de diseño, 162-163
segunda forma normal, 152, 162
tercera forma normal, 152-153
diseño Modelo-Vista-Controlador, 551-552
disparadores
alter, 82
cuándo no utilizarlos, 82-83
DB2 de IBM, 576
disable, 82
drop, 82
en SQL, 81-83
necesidad, 80-81
Oracle, 552-553
PostgreSQL, 548-549
recuperación, 82-83
sintaxis no estándar, 82
SQL Server de Microsoft, 588-589
tablas de transición, 81-82
disparadores de fila, 552-553
disparadores de sentencias, 552-553
disponibilidad
consistencia, 401
copia de seguridad remota, 400
enfoque basado en la mayoría, 399-400
enfoque leer uno, escribir todos, 399-400
robustez, 398
selección de coordinador, 400-401
teorema CAP, 401
distribuidor, 598
división binaria, 424
división cuadrática, 509-510
división horizontal, 375
división recursiva, 249-250
división(es)
asociación, 375-376, 379, 557
atributos, 424
bases de datos distribuidas, 392-393
clasificadores, 424
compuestas, 557-558
computación en la nube, 406-407
condiciones, 424
disponibilidad, 398-401
exploración de una relación, 376
intervalo, 375-376, 378, 557
limpieza, 560
lista, 557
optimización de consultas, 382-383
Oracle, 556-558, 560
referencia, 557
reunión, 379-381
SQL Server de Microsoft, 590
DML procedimentales, 5
Document Object Model (DOM), 176
dominio, 20
dominios atómicos, 20
bases de datos orientadas a objetos, 450
diseño de bases de datos relacionales, 149
primera forma normal, 149
Driver-Manager, clase, 72
drop index, 243
drop table, 29, 74
drop trigger, 82
drop type, 63
durabilidad, 10-11, 46, 292, 294-295
definición, 293
estructura de almacenamiento, 295-296
sistemas de copia de seguridad remota, 353
transacciones distribuidas, 391-392
e-catálogos, 522
Eclipse, 174
eficiencia, 3-4
ejemplar de relación, 20-22, 119
ejemplares, 4, 427
ejemplares de entrenamiento, 422
ejemplos aleatorios, 277
eliminación de duplicados, 261-262
encapsulado, 499-500
encauzamiento, 249-263
bases de datos paralelas, 382-383
dirigido por demanda, 263-264
dirigido por productor, 263-265
encauzamiento dirigido por la demanda, 263-264
encauzamiento dirigido por productor, 263-264
enfoque TF-IDF, 439-440
entorno integrado de desarrollo (IDE), 48, 138, 174, 179, 192, 197, 441
entrada de índice, 218
entropía, medida, 424
envenenamiento del motor de búsqueda, 437-438
envío de elementos, 365
equirreunión, 254-259, 261-262, 264, 379, 384
equivalencia
álgebra relacional, 271-274
análisis de coste, 279-280
ejemplos de transformación, 272-273
ordenación de reunión, 273-274
error del sistema, 335
error lógico, 335
error(es), 78
arquitecturas del sistema, 370
diseño de aplicaciones, 182, 496-497
flujos de trabajo, 521
sistemas de recuperación, 335
transacciones, 517, 522
ERWin, 570
escalado, 365-366
escape, 35
escritura sucia, 302
escrituras externas observables, 296-297
espacio de intercambio, 345
espacio de tabla, 545, 558
espacios de trabajo analítico, 552
especialización disjunta, 133-134
especialización
conjunto de entidad única, 134
modelo entidad-relación (E-R), 133-134
parcial, 135
total, 135
especificación de requisitos funcionales, 7, 117
especificidad, 426
espejo, 200-202, 595-596
esquema de bases de datos. Véase esquemas
esquema de copia en la sombra, 337
esquema de menos usado recientemente (LRU), 212
esquema de redundancia P + Q, 203
esquema de vector de versiones, 513
esquema el más recientemente usado (MRU), 212
esquema de numeración de versiones, 513-514
esquema(s), 4
ajuste de rendimiento, 491-492
álgebra relacional, 97-107
almacén de datos, 420-422
autorización, 66
bloqueos, 309-319
cálculo relacional de tuplas, 107-109
catálogos, 64
combinación, 130-131
conjuntos de entidades débiles, 128-129
conjuntos de entidades fuertes, 128-129
conjuntos de relaciones, 129-130
control de concurrencia, 309-331 (véase también control de concurrencia)
copia en la sombra, 338
cubrimiento canónico, 154-155
dependencias funcionales, 149-157
diseño de bases de datos relacionales, 147-165
documentos XML, 469-472
estructuras de consulta básica de SQL, 29-34

generalización, 134-137
lenguaje de definición de datos (DDL), 27-29
marcas de tiempo, 318-319
mayores, 147-148
menores, 148-149
minería de datos, 422-429
modelo entidad-relación (E-R), 118-141
modelo relacional, 20-23
modificación de la organización física, 13
notaciones alternativas para modelado de datos, 137-140
numeración de versiones, 513-514
reducción a relacional, 128-131
redundancia, 113
sistemas de recuperación, 335-354
estabilidad de cursor, 327
establecer el nivel de aislamiento de instantánea secuenciable, 302
estadísticas
computación, 277
estimación del tamaño de la reunión, 276
información de catálogo, 274-275
mantenimiento, 277
muestras aleatorias, 277
número de valores distintos, 277-278
optimización de consultas, 274-277
selección de la estimación de tamaño, 275-276
estado inconsistente, 295. Véase también consistencia
estados de terminación no aceptables, 520
estados de terminación, 520
Estados Unidos, 8, 21, 118, 120, 185, 370, 403, 409, 436
etiqueta
diseño de aplicaciones, 171, 175, 182
recuperación de información, 433
XML, 465-468, 470, 472, 474, 480
evitar desbordamiento, 259
EXEC SQL, 76-77
ejecutar, 66
exploración compartida, 285
exploración de archivos, 251-252, 255-256, 264
exploración secuencial, 548
exportación en masa, 488
expresiones final/not final, 451-452
Facebook, 14, 405
factoriales, 298
fallos latentes, 204
fallos, 197, 212-213
acciones después de, 341-342
algoritmos para, 341-342
ARIES, 349-352
clasificación de fallos, 335
puntos de comprobación, 340-341
sistemas de recuperación, 341-342 (véase también sistemas de recuperación)
transacciones, 293
falsa eliminación, 439
falsos negativos, 426, 439-440
falsos positivos, 426, 439-440
fase de crecimiento, 311-312
fase de deshacer, 342
fase de diseño físico, 7, 118
fase de diseño lógico, 7, 117-118
fecha actual, 61
fenómeno fantasma, 325-326
firma digital, 187
flujo de tareas. Véase flujo de trabajo
flujo de trabajo, 141-142, 394-395, 480
aplicaciones multisistema, 519
capa de la lógica de negocio, 176-177
ejecución, 519-521
errores, 521
especificación, 519-521
estados de terminación aceptables, 520
estados de terminación no aceptables, 520
fallos, 520-522
pasos, 519
recuperación, 521
rendimiento, 487-496
sistemas de gestión, 521-522
tareas, 519
transaccional, 519-522
variables externas, 520-521
FLWOR (for, let, where, order by, return), expresiones, 473-474
forma normal de Boyce-Codd (BCNF)
algoritmos de descomposición, 157-160
conservación de dependencia, 151-152
diseño de bases de datos relacionales, 151-152
forma normal de clave de dominio (DKNF), 161
forma normal de reunión-proyección (PJNF), 161
formas normales, 8
Boyce-Codd (BCNF), 151-152, 157-160
claves de dominio (DKNF), 161
cuarta, 160, 161
dependencias de reunión, 161
dominios atómicos, 149
primera, 149
quinta, 161
reunión-proyección (PJNF), 161
segunda, 162
superiores, 152-153
tercera, 152-153
tipos de datos complejos, 450
fragmentación, 390-391
fragmentación horizontal, 390
fragmentación vertical, 390
frecuencia inversa de documento (IDF), 434
funciones (véase también funciones de asociación)
construcciones del lenguaje, 79-80
DB2 de IBM, 570-571
declaración, 78
escritura en SQL, 77-80
polimórficas, 537-538
PostgreSQL, 539-540
rutinas de lenguaje externo, 79-80
sintaxis, 77-78, 80
transición de estado, 539
XML, 475
funciones de asociación, 208-209, 217, 244-245
abiertas, 236
actualizaciones, 237-241
borrado, 234, 236-237, 241-242
búsqueda, 237-238, 241-242
cajones insuficientes, 235
cerradas, 236
consultas, 237-241
desbordamiento, 235-236
dinámicas, 237-241
división, 379
estáticas, 234-237, 240-241
estructuras de datos, 237-238
extensibles, 237
índices, 237, 241-242
inserción, 236, 238-241
Oracle, 557
PostgreSQL, 546
reunión, 380-381
funciones definidas por el usuario (UDF), 570-571
ganancia de información, 424
generadores de informes, 180-181
generalización
agregación, 135-136
conjunto subclase, 134
conjunto superclase, 134
definición de condiciones, 135
definición de usuario, 135
diseño de abajo arriba, 134
diseño de arriba abajo, 134
disjunto, 135
herencia de atributos, 134-135
modelo entidad-relación (E-R), 134-135
parcial, 135
representación, 136-137
restricciones, 134-136
superposición, 135
total, 135
Generalized Inverted Index (GIN), 547
Generalized Search Tree (GiST), 546-547
gestión distribuida de bloqueos, 395
gestor de almacenamiento, 9-10
gestor de archivos, 9

gestor de bloqueo único, 395-396
gestor de integridad, 9
gestor de recuperación, 10-11
gestor de recursos, 519
gestor de transacciones, 9-10, 391-392
GET, método, 183
getColumnCount, método, 74
getConnection, método, 72
getString, método, 72
Gini, medida, 424
Glassfish, 174
Google, 14
bases de datos distribuidas, 405, 407-408
diseño de aplicaciones, 171-172, 179, 184
PageRank, 436-438
recuperación de información, 441 (véase también recuperación de información)
grado de salida, 222
grafo, 313-314
grafo de espera, 315-316, 398
grafo de espera global, 398
grafo de espera local, 398
grafo de precedencia, 300
granularidad múltiple
arquitectura en árbol, 316-318
control de concurrencia, 316-318
definición jerárquica, 316
modelo de intención compartida, 317
modelo de intención exclusiva, 317
modelos de intención exclusiva y compartida, 317
protocolo de bloqueo, 317-318
Greenplum, 383
guiones del lado del cliente, 175-176
guiones del lado del servidor, 174-175
hackers. Véase seguridad
hardware, ajuste, 490-491
hardware RAID, 204
herencia, 134-135
método de sobrescritura, 452
SQL and, 451-453
tablas, 452-453
tipos, 452
tipos estructurados, 452
herencia de atributos, 134-135
herencia de tabla, 452-453
heurísticas, 509-510
análisis de datos, 425, 429
ávidas, 429
bases de datos distribuidas, 404
bases de datos paralelas, 383
DB2 de IBM, 577
optimización de consultas, 278-281, 285-286
Oracle, 560
recuperación de información, 442
SQL Server de Microsoft, 592
Hibernate, sistema, 177-178
HIPAA, 596
hiperenlace
clasificación por popularidad, 435-436
envenenamiento del motor de búsqueda, 437-438
PageRank, 436-438
histogramas, 87, 274-276, 286, 376, 425, 548, 559, 577, 592
HITS, algoritmo, 438
hojas de estilo, 171
hojas de estilo en cascada (CSS), norma, 171
homónimo, 438-439
HP-UX, 569
IBM AIX, 569
IBM MVS, 569
IBM OS/400, 569-570
IBM VM, 569-570
IBM z/OS, 569
ID de tupla, 546
identificación de relaciones, 126
identificador global de clase, 499
identificador global de compañía, 499
identificador global de producto, 499
identificadores, 253
etiquetas, 465-466
global, 499
normas, 499-500
OrdPath, 602-603
Illustra Information Technologies, 535-536
inanición, 316
incompletitud, 118
independencia de datos física, 3
índices, 9, 62, 244
acceso con clave múltiple, 221, 233-234
actualización, 221
agrupamiento, 217-218, 220-221, 251
ajuste de rendimiento, 491-493
árboles, 546-547 (véase también árboles)
asociativos, 217 (véase también funciones asociativas)
basados en funciones, 555-556
bloqueo, 574
búsqueda lineal, 251
carga en masa, 231-232
clases de operadores, 547
clave de búsqueda, 217
cobertura, 234
comparaciones, 252
compuestos, 252-253
construcción, 547-548
control de concurrencia, 328-329
datos espaciales, 508-510
DB2 de IBM, 571-574
definición en SQL, 243-244
densos, 218-220
dispersos, 218-220
divisiones, 556-557
dominio, 556-557
en expresiones, 547
identificadores de fila lógicos, 554-555
identificadores, 253
Índices invertidos generalizados (GIN), 547
invertidos, 439-440
lenguajes de programación persistentes, 457-461
mapas de bits, 233-234, 242-244, 247, 255-256
multicolumna, 547
multinivel, 219-220
no agrupados, 218
operador de selección, 251-252
Oracle, 553-558
ordenación, 253-254
ordenados, 217-222, 241-242
parciales, 547
PostgreSQL, 540, 545-548
primarios, 217-218, 251-252
procesamiento de consultas, 251-252
punteros, 253
recuperación de información, 439-440
reubicación de registro, 231
reunión, 556 (véase también reuniones)
rutinas de soporte, 540
secuenciales, 221-222
secundarios, 218, 220-221, 231, 251-253
sobrecarga de espacio, 217, 219, 222, 241
SQL Server de Microsoft, 588-590
tiempo de acceso, 217, 219, 241
tiempo de borrado, 217, 220, 225, 227-230, 241
tiempo de inserción, 217, 220, 225-227, 229-230
tipos de acceso, 217
únicos, 547
vistas materializadas, 284
XML, 552
información de usuario final, 184
información empresarial, 1
información topográfica, 507
Ingres, 14
inserción, 29, 44-45
asociación, 236, 238-241
búsqueda, 328
control de concurrencia, 324-326
EXEC SQL, 77
fenómeno fantasma, 325-326
PostgreSQL, 538
privilegios, 64-65
sentencias preparadas, 72-74
transacciones, 294, 304
vistas, 56-57
inserción en masa, 488
instantánea
comandos DML, 541-542
Control de concurrencia multiversión (MVCC), 541-545
lectura comprometida, 593

PostgreSQL, 541-545
SQL Server de Microsoft, 593
instantánea consistente con transacciones, 397
integridad referencial, 5, 23-24, 59-61, 67, 81, 293
intercambio en caliente, 205
Interconexión de pequeños sistemas de computadora (SCSI), 197
interfaces de usuario, 13
almacenamiento, 197 (véase también almacenamiento)
arquitectura cliente-servidor, 15, 90, 169-170, 352-362, 365, 370, 372
arquitecturas de aplicación, 176-178
capa de acceso a los datos, 176-178
capa de la lógica de negocio, 176-177
capa de presentación, 176
como componente de *back-end*, 169
computación en la nube, 405-409
cookies, 172-174, 182-183
CRUD, 180
DB2 de IBM, 570
generadores de informes, 180-181
guiones del lado del cliente, 175-176
herramientas para la construcción, 179-180
interfaz de pasarela común (CGI), 171-172
lenguajes de programación persistente, 459
móvil, 511-514
operación desconectada, 178-179
PostgreSQL, 535-537
programas de aplicación, 169-170
Protocolo de transferencia de hipertexto (HTTP), 170-173, 178, 182-183, 188
seguridad, 181-182
servicios Web, 178
World Wide Web, 170-172
Interfaz de nivel de llamada (CLI, Call Level Interface), 498
Interfaz de pasarela común (CGI), norma, 171-172
Interfaz de programación del servidor (SPI), 540
Interfaz Fibre Channel, 197-198
Interfaz FireWire, 197
interferencia, 366
Internet, 14
acceso directo de usuario, 1
inalámbrica, 512-513
intersección, 24
intersección de conjuntos, 587
intersect, 37, 271
intersect all, 37
intervalo de recuperación, 594-595
intervalo(s), 504
is unknown, 38
iteración, 79, 84-85
J++, 587, 599
Jakarta, proyecto, 174
Java, 6, 71, 76-77, 175, 449
DOM API, 476
JDBC, 71-75
Lenguaje unificado de modelado (UML), 139
metadatos, 74-75
sistemas persistentes, 460-461
SQLJ, 77
Java 2 Enterprise Edition (J2EE), 174, 551-552
Java Database Objects (JDO), 460
Java Server Pages (JSP)
aplicaciones Web, 180
diseño de aplicaciones, 170, 173-176
guiones del lado del cliente, 175-176
guiones del lado del servidor, 174-175
seguridad, 83
servlets, 173-176
JavaScript
diseño de aplicaciones, 175-176, 179
Representation State Transfer (REST), 178
seguridad, 181-185
JavaScript Object Notation (JSON), 178, 405-406
JavaServer Faces (JSF), 179
JBoss, 174, 180
JDBC (Java Database Connectivity), 171, 498, 549
caché, 181
características de metadatos, 74-75
columna blob, 74
columna clob, 74
conexión a la base de datos, 71-72
conjunto de resultados actualizable, 74
envío de sentencias SQL, 72
modelo E-R, 121, 124
protocolo de información, 71-72
recuperación de resultados de consultas, 72-73
sentencias llamables, 74
sentencias preparadas, 72-74
SQL avanzado, 71-72
JPEG (Joint Picture Experts Group), 510
kernel, funciones, 425-426
Language Integrated Querying (LINQ), 499, 597
LDAP Data Interchange Format (LDIF), 410
lectura adelantada, 198
lectura comprometida
desarrollo de aplicaciones, 493
gestor de transacciones, 302, 306, 319, 326-327
Oracle, 562
PostgreSQL, 541-543
SQL Server de Microsoft, 593
lectura de predicados, 324-326
lectura fantasma, 541-543, 579-580
lectura no comprometida, 302
lectura no repetible, 541
lectura repetible, 302
lectura sucia, 541, 562
lenguaje de definición de datos (DDL), 5-6, 15
almacenamiento, 27
autorización, 27
conjunto de relaciones, 27-29
consulta, 9-10
control de concurrencia, 544-545
DB2 de IBM, 570-571, 574
definición de esquema, 13, 27-29
índices and, 27
integridad, 27
Oracle, 553, 562
PostgreSQL, 544-545, 547
principios de SQL, 27-29, 46
seguridad, 27
SQL Server de Microsoft, 586-589, 595, 599
tipos básicos, 28
volcado, 345-346
Lenguaje de descripción de interfaces (IDL), 498-499
lenguaje de manipulación de datos (DML), 6-7
aislamiento de instantánea, 545 (véase también aislamiento de instantánea)
autorización, 64
compilador, 9-10
consultas, 9-10
control de concurrencia, 541-542, 544
declarativo, 5
definido, 5, 15
disparadores, 552-553
gestor de almacenamiento, 9-10
lenguaje anfitrión, 7
Oracle, 552-553, 555, 562
PostgreSQL, 541
precompilador, 7
procedimental/no procedimental, 5
SQL Server de Microsoft, 588-589, 595, 603
Lenguaje de marcado de realidad virtual (VRML, Virtual Reality Markup Language), 176-177
Lenguaje de marcado generalizado estandarizado (SGML), 465
Lenguaje de marcas ColdFusion (CFML), 174
Lenguaje de marcas de hipertexto (HTML), 171
DataGrid, 179-180
desarrollo rápido de aplicaciones (RAD), 179
embebido, 179
entornos de aplicación Web, 179-180

guiones del lado del cliente, 175-176
guiones del lado del servidor, 174-175
hojas de estilo, 171
Java Server Pages (JSP), 174-176
recuperación de información, 433 (véase también recuperación de información)
seguridad, 181-188
sesiones web, 171-172
XML, 465
Lenguaje unificado de modelado (UML), 8
asociación, 139-140
componentes, 139
conjuntos de relaciones, 139-140
restricciones de cardinalidad, 139-140
lenguajes de consulta, 110. Véase también cada lenguaje específico
acceso desde un lenguaje de programación, 71-77
álgebra relacional, 97-107
cálculo relacional de dominios, 109-110
cálculo relacional de tuplas, 107-109
formal relacional, 97-110
modelo relacional, 23-24
no procedimental, 107-109
potencia expresiva de los lenguajes, 108, 110
procedimental, 97-107
sistemas centralizados, 361-362
temporal, 504
Lenguaje de descripción de servicios Web (WSDL, Web Services Description Language), 480
lenguajes de guiones, 175
lenguajes de marcado. Véase también cada lenguaje específico
estructura, 465-469
etiquetas, 465-466
materialización, 262-263
procesamiento de archivos, 465
replicación maestro-esclavo, 397
tabla maestra, 488-489
transacciones, 465-466
lenguajes de programación persistente, 461
actualizaciones, 459
alcanzabilidad, 460
bases de datos basadas en objetos, 456-461
C++, 458-460
correspondencia de bases de datos, 460
definición, 457
enfoque, 467-468
extensiones de clases, 459-460
identidad de objetos, 458
interfaz iterator, 459
Java, 460-461
mejora de los códigos de byte, 460
objetos persistentes, 459
persistencia de objetos, 457-459
punteros, 458-460
relaciones, 459
sobrecarga, 459
tipos de referencia única, 460
transacciones, 459
lenguajes de programación. Véase también cada lenguaje específico
acceso a SQL, 71-77
operaciones sobre variables, 71
lenguajes no procedimentales, 23
lenguajes procedimentales, 9
DB2 de IBM, 550
modelo relacional, 23
Oracle, 552, 567
PostgreSQL, 538-540
SQL avanzado, 71-72, 77, 79
Linux, 535, 569, 577
llamada, 519
llamada a procedimiento remoto (RPC), mecanismo, 519
llamada hacia atrás, 364
localizadores uniformes de recursos (URL), 170-171
lógica de negocio, 12, 77
DB2 de IBM, 581-582
diseño de aplicaciones, 169, 173, 176-177, 179, 185, 188
Oracle, 551
SQL Server de Microsoft, 587, 589, 599, 603-605
mantenimiento de vistas, 282-284
mantenimiento incremental de vistas, 282-284
mapa de bits de existencia, 242-243
mapas de bits, índices, 233-234, 244, 247
mapas de bits nulos, 207
máquina universal de Turing, 6
Máquinas de vectores de soporte (SVM), 425-426, 567
máquinas virtuales, 365
marca de tiempo local, 61
marca(s) de tiempo, 60-61
bases de datos distribuidas, 396-397
con zona de tiempo, 504
contador lógico, 318
control de concurrencia, 318-319, 327
datos temporales, 504
esquemas de ordenación, 318-319
esquemas multiversión, 321-322
regla de escritura de Thomas, 319-320
transacciones de larga duración, 526
transacciones, 303-304
vuelta atrás, 319-320
máximo, 38, 43, 105, 262-263
mediadores, 404, 480-481
mejores particiones, 424-425
memoria. Véase también almacenamiento
.NET Common Language Runtime (CLR), 600
acceso a datos, 336-337
acceso aleatorio no volátil (NV-RAM), 199
banco compartido, 563
bases de datos en memoria principal, 523-525
caché, 195, 383-384 (véase también caché)
carga en masa de índices, 231-232
coste de consulta, 252
desbordamiento, 259
disco magnético, 195-200
estructuras de Oracle, 563-564
flash, 182, 195, 199, 200-201, 233
lenguajes de programación persistentes, 456-461
memoria intermedia de registro de rehacer, 563
memoria intermedia, 563 (véase también memoria intermedia)
multitarea, 517-519
óptica, 195
principal, 194-195
salida forzada, 336-337
sistemas de recuperación, 336-337 (véase también sistemas de recuperación)
SQL Server de Microsoft, 595-596
memoria de acceso aleatorio no volátil (NVRAM), 199
memoria flash NAND, 195, 200-201
memoria flash NOR, 195, 199
memoria intermedia de escritura no volátil, 199
memoria intermedia
bancos múltiples, 581
bases de datos en memoria principal, 524-525
bases de datos y, 343-344
DB2 de IBM, 572-573
directivas de forzar/no forzar, 343-344
estructura de archivos, 198-199
forzar salida, 336-337
gestión, 343-345
políticas de apropiación/no apropiación, 343
políticas de sustitución, 212-213
registros de registro, 343
regla de registros de escritura adelantada (WAL), 343-344
rol del sistema operativo, 344-345
servidores de transacciones, 363
sistemas de recuperación, 343-345
memoria intermedia, gestor de, 9, 211-212
menos, 37
mensajería persistente, 393-394
metadatos, 6, 74-75
mezcla
ajuste de rendimiento, 489
compleja, 558-559
eliminación de duplicados, 261

Oracle, 558-559
ordenación por mezcla externa paralela, 378
procesamiento de consultas, 253-254, 256-258
Microsoft, 2, 14, 63
almacenamiento, 199
bases de datos distribuidas, 405
bases de datos paralelas, 383
diseño de aplicaciones, 174, 178-181, 183-184
optimización de consultas, 284
SQL avanzado, 72, 76-77, 80, 82, 88, 91
Microsoft Active Server Pages (ASP), 179
Microsoft Database Tuning Assistant, 492
Microsoft Distributed Transaction Coordinator (MS DTC), 593
Microsoft Office, 26, 180, 479
Microsoft Transaction Server, 517
Microsoft Windows, 87, 192, 511
almacenamiento, 199
DB2 de IBM, 569-570, 577
PostgreSQL, 535, 549
SQL Server, 585-587, 593, 595-597
migración de aplicaciones, 497
minería de datos, 12, 15, 362, 419-420
agrupamientos, 422, 427-429
clasificación, 422-427
definición, 422
ganancia de información, 424
medida de entropía, 424
medida de Gini, 424
mejores divisiones, 424-425
Oracle, 567
patrones descriptivos, 422
predicción, 422-427
pureza, 424
reglas de asociación, 427-428
reglas, 422
SQL Server de Microsoft, 605
texto, 428
visualización de datos, 429
mínimo, 38-39, 105, 262-263
minpctused, 573
modelo de datos de objetos relacional, 12
modelo de datos jerárquico, 4
modelo de datos orientado a objetos, 12
modelo de datos simiestructurado, 4, 12
modelo de espacio de vector, 435
modelo de objetos componentes (COM), 599
modelo de paseo aleatorio, 436
modelo de servidor único, 517-518
modelo entidad datos, 479
modelo entidad-relación (E-R), 4, 8, 117, 142, 456
agregación, 135-136
atributos, 118, 120-121, 131-135, 149
características extendidas, 133-137
conjuntos de entidades, 118-120, 123-124, 125-129, 131-133
conjuntos de relaciones, 119-120, 129-131, 133-134
dominios atómicos, 149
elementos de diseño, 131-133
especialización, 133-134
esquema de empresa, 118
generalización, 134-137
Lenguaje unificado de modelado (UML), 139-140
modelo de datos orientado a objetos, 12
normalización, 162
notaciones alternativas de modelado de datos, 137-140
reducción a esquemas relacionales, 128-131
redundancia, 123-124
restricciones, 121-123
tipos de datos complejos, 449-450
modelo muchos servidores, muchos encaminadores, 518
modelo muchos servidores, un encaminador, 517
modelo relacional, 4
claves, 21-22
desventajas, 14
dominio, 20
esquema, 20-23, 136-137, 477
estructura, 19-20
lenguajes de consulta, 23-24
operaciones, 23-25
relación referenciante, 22
reunión natural, 23-24
tablas para, 19-21, 23-24, 90-91
tuplas, 19-20, 23-24
modelos de datos. Véase modelos específicos
modo compartido y exclusivo, 317
monitor del procesamiento de transacciones, 517
arquitectura, 517-519
cola durable, 518
coordinación de aplicación usando, 518-519
facilidades de presentación, 518
intercambio, 517
modelo de un único servidor, 517-518
modelo muchos servidores, muchos encaminadores, 518
modelo muchos servidores, un encaminador, 517
multitarea, 517-519
motor de ejecución de consultas, 249
movilidad, 503, 514
actualizaciones, 513-514
comunicación inalámbrica, 511-513
consistencia, 513-514
consultas, 512
difusión de datos, 512-513
encaminamiento, 512
esquemas de números de versión, 513-514
modelo de computación móvil, 511-513
recuperabilidad, 513
reportes de invalidación, 513
MPEG (Moving Picture Experts Group), 510-511
multihebra, 383-384, 517-518
multiprocesadores simétricos (SMP), 569
multitarea, 362, 517-519
negación, 276
NetBeans, 174, 179
niveles de conformidad, 75-76
no anidado, 454-455
no repudio, 187
nodos intermedios, 222
nodos internos, 222
nodos. Véase también almacenamiento
actualizaciones, 225-229
árboles B^+, 222-232
arquitectura en malla, 266
DB2 de IBM, 572
división, 225, 328
granularidad múltiple, 316-318
sistemas distribuidos, 368
sobreajuste, 425
XML, 472
política de no forzar, 343-344
Nombre distinguido (DN), 410
Nombres distinguidos relativos (RDN), 410
normalización, 7-9
desnormalización, 163-164
diseño de bases de datos relacionales, 162
modelo entidad-relación (E-R), 162
rendimiento, 163-164
política de no apropiación, 343
normas X/Open XA, 498
normas
ANSI, 27, 497
anticipatorias, 497
conectividad de base de datos, 498
DBTG CODASYL, 498
Interfaz del nivel de llamada (CLI, Call Level Interface), 498
ISO, 27, 410, 497
ODBC, 498-499
reaccionaria, 497
SQL, 498
Wi-Max, 512
X/Open XA, 498
XML, 499
not exists, 41, 85

not in, 40-41
not null, 28, 37, 58-60, 63
not unique, 42
núcleos, 361
numérico, 28-29
Número de cambio del sistema (SCN), 561-562
Número de secuencia del registro (LSN), 349-351
nvarchar, 28
objeto ResultSet, 71-72, 74-75, 177, 179-180, 224
objeto Statement, 72-74
ODBC (Open Database Connectivity), 171, 498
caché, 181
definición de la API, 74-75
definiciones de tipos, 75
niveles de conformidad, 75-76
normas, 498
PostgreSQL, 549
SQL avanzado, 74-76
SQL Server de Microsoft, 597
OLAP (Procesamiento analítico en línea), 495
aplicaciones, 88-90
atributo all, 89-91
cláusula order by, 91
cláusula pivot, 91, 93
cubo de datos, 89, 91-93
datos multidimensionales, 88
división, 89
en SQL, 91-92
función decode, 92
implementación, 90
Oracle, 552
particularización, 89
SQL Server de Microsoft, 585, 605
tablas relacionales, 90-92
valor null, 90
OLAP híbrido (HOLAP), 90
OLE-DB, 597
OLTP (Procesamiento de transacciones en línea), 495-496, 555, 564, 604
correspondencia uno a muchos, 121, 124
correspondencia uno a uno, 121, 124
on condition, 51-52
on delete cascade, 60, 82
ontologías, 438-439
OOXML (Office Open XML), 479
opción de concesión, 66
opción with check, 57
Open Document Format (ODF), 479
operación desconectada, 178
operación deshacer lógica, 346-348
operación or, 37-38
operación proyección, 98
operación rename, 34-35, 101-102
operación request
aislamiento de instantánea, 323
bloqueos, 309-313, 315-317, 330
búsqueda, 328
esquemas multiversión, 322
granularidad múltiple, 316-317
manejo de bloqueo, 315-316
marcas de tiempo, 318
operación XOR, 186
operación y, 30, 37-38, 559
operaciones de cadena
agregado, 38
escape, 35
especificación de atributos, 35
función upper, 35
JDBC, 71-75
like, 35
menor, 35
orden para mostrar las tuplas, 35-36
predicados where, 36
recuperación del resultado de consulta, 72-73
similar a, 35
trim, 35
operaciones de conjuntos, 36-37
comparación de conjuntos, 41-42
DB2 de IBM, 576
intersección, 24, 37, 271, 455
optimización de consultas, 277
procesamiento de consultas, 261-262
subconsultas anidadas, 40-42
unión, 36-37, 98-99, 153, 271
operaciones de lectura/escritura, 304-305
operaciones lógicas
consistencia, 346
liberación temprana, 345-348
registros de registro de deshacer, 346-348
vuelta atrás, 346-348
optimización de consultas, 10, 249, 255-256, 260, 286
actualizaciones, 284-285
agregación, 277
álgebra relacional, 269-274
análisis de coste, 269-270, 274-279
bases de datos distribuidas, 402
bases de datos paralelas, 382-383
búsqueda parcial, 592
caché de resultados, 561
DB2 de IBM, 577
divisiones, 559-560
ejecución paralela, 560-561
elección de plan, 278-282
equivalencia, 270-273
estimación de estadísticas de los resultados de expresión, 274-277
estructura de proceso, 561
exploración compartida, 285
heurísticas, 279-281
mejores-K, 284
minimización de reunión, 285
multiconsulta, 285
nested subconsultas, 281-282
operaciones de conjuntos, 277
Oracle, 558-560
paramétricas, 285
PostgreSQL, 547-549
selección de la ruta de acceso, 559-560
simplificación, 591-592
SQL Server de Microsoft, 590-593
SQL Tuning Advisor, 558-559
transformaciones, 271-274, 558-559
vistas materializadas, 282-284
optimización de los mejores K, 284
optimización multiconsulta, 285
optimización paramétrica de consultas, 285
Oracle, 2, 14, 96, 535
actualizaciones, 561-562
agrupamientos, 558, 564
almacén de datos, 551
árboles, 567
archivador, 564
arquitectura del sistema, 374, 377, 397, 563-565
asociación, 557
base de datos privada virtual, 555
caché, 561-562, 564
cifrado, 554-555
compresión, 554
control de concurrencia, 561-563
datos externos, 565-566
diseño de base de datos, 160, 174, 179, 184-185, 551-552
disparadores, 553
distribución, 565-566
ejecución paralela, 560
escritor de base de datos, 564
escritor del registro, 564
espacio de trabajo analítico, 553
estructuras de memoria, 563-564
estructuras de proceso, 564-565
Exadata, 565-566
guarda de datos, 563
herramientas de administración de bases de datos, 566-567
índices, 553-558
minería de datos, 567
modelado dimensional, 552, 557
monitor del sistema, 564
monitores de proceso, 564
niveles de aislamiento, 562-563
optimización de consultas, 270, 277, 280, 284, 558-560
optimizador, 559-560
particiones, 556-557, 560
principios de SQL, 26, 34, 37, 43, 63, 72, 77-78, 80, 82, 88, 91
procesamiento de consultas, 551-552, 553-558
proyección, 565

punto de comprobación, 564
Real Application Clusters (RAC), 564
recuperación, 561-564
replicación, 565-566
reuniones, 556, 565
secuencialidad, 562-563
segmentos, 553
seguridad, 554-555
selección de ruta de acceso, 559
servidor compartido, 564
servidores dedicados, 563-564
SQL Loader and, 566
SQL Plan Management, 560-561
SQL Tuning Advisor, 560
tablas, 553-555, 558, 565-566
transacciones, 302, 304, 322-324, 330, 334
transformaciones, 558-559
variaciones de SQL, 552-553
vistas materializadas, 557-559, 565
XML DB, 552
Oracle Application Development Framework (ADF), 551-552
Oracle Automatic Storage Manager, 565
Oracle Automatic Workload Repository (AWR), 566
Oracle Business Intelligence Suite (OBI), 551
Oracle Database Resource Management, 566-567
Oracle Designer, 551
Oracle Enterprise Manager (OEM), 566
Oracle JDeveloper, 551
Oracle Tuxedo, 517
ordenación, 253
algoritmo de ordenación por mezcla externa, 253-254
análisis de coste, 254
división de intervalos, 378
eliminación de duplicados, 260-261
por mezcla externa paralela, 378
PostgreSQL, 548
topológica, 300-301
XML, 554
ordenación por división de intervalos, 378
ordenación por mezcla externa paralela, 378
ordenación topológica, 300-301
organización de archivos, 2. Véase también almacenamiento
agrupación multitabla, 208-210
árboles B^+, 231
asociación, 208
blob, 62, 74, 208, 231, 478, 571-572, 602
clob, 62, 74, 208, 231, 476-478, 570-571
estructura del sistema, 206
estructurado, 206-208
indexado, 217 (véase también índices)
punteros, 206
registros de longitud variable, 206-208
registros de tamaño fijo, 206-207
secuencial, 208-209
seguridad, 2-3 (véase también seguridad)
tiempo de acceso a bloque, 199
organización de archivos de agrupación multitabla, 208-210
Organización internacional de normalización (ISO), 27, 410, 497
organizar por dimensiones, 574
PageLSN, 349-351
PageRank, 436-439
paginación en la sombra, 338
palabras clave
clasificación, 433-438
envenenamiento del motor de búsqueda, 437-438
homónimas, 438-439
índices, 439
ontologías, 438-439
palabras de parada, 434
PostgreSQL, 538
simplificación de consultas, 591-592
sinónimos, 438-439
tipos de datos complejos, 450-451
palabras de parada, 434
paralelismo(s), 201-202
paralelismo de datos, 378
paralelismo de E/S
asociación, 376
esquema de intervalos, 376
esquema de reparto, 376
manejos de desvíos, 376-377
técnicas de división, 375-376
paralelismo de grano fino, 362
paralelismo independiente, 382
paralelismo interoperación, 378, 382
paralelismo intraconsulta, 377-378
paralelismo intraoperación
agregación, 381
coste de evaluación de operación, 381
eliminación de duplicados, 381
grado de paralelismo, 378
ordenación paralela, 378-379
ordenación por división de rangos, 378
ordenación por mezcla externa paralela, 378
proyección, 381
reunión paralela, 378-381
selección, 381
paralelismos interconsulta, 377-378
Partner Interface Processes (PIP), 499
paso de deshacer, 351
pasos, 519
PATA (ATA paralelo), 197
patrones descriptivos, 422
pctfree, 573
Perl, 80, 174, 549
PHP, 174-175
PL/SQL, 77, 80
plan de ejecución de consultas, 249
planificación(es)
almacenamiento, 198
optimización de consultas, 382-383
PostgreSQL, 537
SQL Server de Microsoft, 599-600
transacciones, 299, 520-521, 525
planificación del brazo de disco, 198
planificaciones secuenciables, 298
planificaciones sin cascada, 301-302
plazos, 525-526
política de apropiación, 343
PostgreSQL, 14, 535
actualizaciones, 538, 542-546
agregación, 548
ajuste de rendimiento, 493
almacenamiento, 545-547
árboles, 546-547
arquitectura del sistema, 549-550
asociación, 546
bases de datos paralelas, 383-384
bloqueos, 543-545
catálogos del sistema, 538
clases de operadores, 547
comandos DML, 541-542
control de concurrencia, 322, 324, 326, 541-545
control de concurrencia multiversión (MVCC), 541-545
disparadores, 548-549
edición en línea de comandos, 535
extensibilidad, 538
funciones, 539-540
Generalized Inverted Index (GIN), 547
Generalized Search Tree (GiST), 546-547
gestión de transacciones, 302, 304, 541-545
ID de tupla, 545-546
índices, 540, 545-547
interfaces de usuario, 585-586
interfaz de programación de servidor, 540
lenguajes confiables/no confiables, 540
lenguajes procedimentales, 540
métodos de acceso, 548
niveles de aislamiento, 541, 543
optimización de consultas, 271, 277
ordenación, 548
principales versiones, 585-586
principios de SQL, 63, 72, 77, 80, 82
procesamiento de consultas, 547-549
punteros, 539, 545-546
recuperación, 334
reglas, 538
restricciones, 538, 548-549
precisión, 426
predicados verdaderos, 31
predicción
clasificadores, 422-427

minería de datos, 422-427
reuniones, 605
primer compromiso gana, 322-323
primera actualización gana, 323
primitivas de evaluación, 249
privacidad, 181, 185, 188, 390, 409, 523
privilegios
all (todos), 64-65
concesión, 64-65
de ejecución, 66
públicos, 64
revocación, 64-65, 67
transferencia, 66-67
procedimientos
construcciones del lenguaje, 79-80
declaración, 78
escritura SQL, 77-80
rutinas de lenguaje externas, 79-80
sintaxis, 77-78, 80
procesador virtual, 376
procesadores multinúcleo, 383-384
procesamiento de consultas, 9, 10, 14-15
agregación, 262
álgebra relacional, 249-250
análisis de coste, 250-252, 254, 258, 260
análisis, 249-250, 265, 590-591
asociación, 257-260
bases de datos distribuidas, 402-404
comparaciones, 251-253
compilación, 590-591
DB2 de IBM, 575-579
disparadores, 548-549
distribuida heterogénea, 597-598
división recursiva, 249-250
eliminación de duplicados, 260-261
encauzamiento, 263-265
evaluación de expresiones, 262-265
evaluación de operación, 240-250
exploración de archivos, 251-252, 255-256, 264
identificadores, 253
LINQ, 597
materialización, 262-263, 577-578
módulo ejecutor, 548-549
móvil, 512
operación de reunión, 254-262
operación de selección, 251-253
operaciones de conjuntos, 261-262
Oracle, 551-552, 558-561
ordenación, 253-254
pasos básicos, 249
planeador estándar, 548
PostgreSQL, 547-549
proyección, 260-261
recuperación de información, 433-443
reordenación, 591-592
sintaxis, 249
SQL Server de Microsoft, 585-588, 591-593, 597-598
SQL, 249
transformación, 402
velocidad de CPU, 250
XML, 602
procesamiento paralelo, 181-182
proceso de envío de mensajes, 394
proceso de escritura del registro, 363-364
proceso de escritura en base de datos, 363
propagación perezosa, 397, 408
Protocolo de acceso a directorio ligero (LDAP), 183, 410-413
Protocolo de acceso simple a objetos (SOAP), 479-480, 499, 597-598
Protocolo de aplicación inalámbrica (WAP, Wireless application protocol), 512-513
Protocolo de compromiso de dos fases (2PC), 369-370, 392-394
Protocolo de compromiso en tres fases (3PC), 389
protocolo de consenso por quorum, 396-397
protocolo de leer uno, escribir todos disponibles, 400
protocolo de leer uno, escribir todos, 400
protocolo de mayoría, 395-396, 399-400
Protocolo de transferencia de hipertexto (HTTP)
ataques de hombre en medio, 183
certificados digitales, 188
diseño de aplicaciones, 170-173, 178, 182-183, 188
Protocolo de acceso simple a objetos (SOAP), 480, 597
Representation State Transfer (REST), 178
sin conexión, 172
Protocolo Secure Electronic Transaction (SET), 523
protocolos basados en grafos, 313-314
protocolos de bloqueo, 311
basados en grafos, 313-314
consenso por quorum, 396
copia primaria, 395
de dos fases, 311-312
gestor de bloqueo distribuido, 395
gestor de bloqueo único, 395
gestor de bloqueos, 312-313, 363
marcado de tiempo, 396-397
por mayoría, 395-396
protocolos de compromiso, 392-395
proyección
consultas, 261, 277
mantenimiento de vistas, 283
Oracle, 525
paralelismo intraoperación, 381
proyección generalizada, 104
proyecto Cyc, 438-439
pruebas de rendimiento
clases de aplicaciones de bases de datos, 495
conjuntos de tareas, 494-495
Transaction Processing Performance Council (TPC), 495-496
publicación, 478, 598-599
punteros. Véase también índices
almacenamiento, 200, 206-210
bases de datos en memoria principal, 524
bases de datos multimedia, 508
control de concurrencia, 328-329
DB2 de IBM, 571, 573
diseño de aplicaciones, 184
lenguajes de programación persistente, 458-459, 461
nodos hijo, 508
optimización de consultas, 284
Oracle, 554
PostgreSQL, 60, 66, 545-546
principios de SQL, 74, 79-80
procesamiento de consultas, 251-253, 256
recuperación de información, 443
sistemas de recuperación, 338, 351
puntos de comprobación
difusos, 344-345, 349
Oracle, 564
servidores de transacciones, 363
sistema de recuperación, 340-341, 344-345
SQL Server de Microsoft, 595
puntos de comprobación difusos, 344-345, 349
pureza, 424
Python, 80, 170, 174, 535-536, 540
QBE, 19, 109, 361
quadtrees, 506, 508-509
quinta forma normal, 161
quorum de escritura, 396
quorum de lectura, 396
RAID (Arrays redundantes de discos independientes), 198, 353, 545
ajuste de rendimiento, 490-491
bit de paridad, 202-203
confiabilidad del rendimiento, 201-202
división a nivel de bit, 201-202
elementos hardware, 204-205
espejo, 200-202
intercambio en caliente, 204-205
mejora de fiabilidad, 200-201
niveles, 202-204
Organización del código de corrección de errores (ECC), 202-203
paralelismo, 201-202
sistemas de recuperación, 336
software RAID, 204
Rational Rose, 570
real, precisión doble, 28

recuperación basada en la similitud, 435, 511
recuperación de información, 12, 415, 444
aplicaciones de, 433-434, 441-442
basada en similitud, 435, 511
buscadores Web, 440
categorías, 442-443
clasificación por relevancia usando términos, 434-435
consulta de datos estructurados, 442
definición, 433
desarrollo del campo, 433
directorios, 442-443
diversidad de resultados, 441
diversidad de resultados de consulta, 441
enfoque TF-IDF, 434-438
envenenamiento de motor de búsqueda, 437-438
extracción de información, 441-442
falsos negativos, 439-440
falsos positivos, 439-440
homónimos, 438
indexado de documentos, 439
medida de eficacia, 439-440
ontologías, 438
PageRank, 436-437
palabras de parada, 434
palabras clave, 433-439
precisión, 439-440
prueba de adyacencia, 436-437
relevancia usando hiperenlaces, 435-438
respuesta a preguntas, 441-442
sinónimos, 438-439
recuperación, 9-10, 62, 428, 511, 520
almacenamiento, 198-199, 202 (véase también almacenamiento)
bases de datos basadas en objetos, 457, 459-461
buscadores Web, 440-441
DB2 de IBM, 571, 576-577, 581
diseño de aplicaciones, 175-180, 487, 491
PostgreSQL, 541, 545, 547-548
recuperación de la información, 435, 439, 443 (véase también recuperación de la información)
SQL avanzado, 72, 74-77, 79-80, 86
SQL Server de Microsoft, 593, 598
redes
área local (LAN), 370-371, 512
modelo de datos, 4, 511-513
movilidad, 511-514
tipos de área extensa (WAN), 370-371
Redes de área extensa (WAN), 370-371
Redes de área local (LAN), 370-371, 512
redes interconectadas, 366-367
redundancia, 2, 118, 123
referencia de relación, 22
referencia new row as, 81
referencia new table as, 81
referencia old row as, 80
referencia old table as, 81
referenciantes, 412
referencias, 59-60, 66
Registro de control de espacio libre (FSCR), 573
Registro de escritura adelantada (WAL), 343-344, 544-545
registro de operación, 528
registro lógico, 346-347, 528
registros de índices, 218
registros del registro de comprobación, 349
registros del registro
ARIES, 352
deshacer, 338-340, 346-347
físicos, 346
identificadores, 338
memoria intermedia, 343
política de apropiación/no apropiación, 343
Registros del registro de compensación (CLR), 349, 351
Regla de registro de escritura adelantada (WAL), 343-344
rehacer, 338-341
sistemas de recuperación, 337-338, 339-341
valores antiguos/nuevos, 337-338
regla de aumento, 153
regla de escritura de Thomas, 319
regla de pseudotransitividad, 153
regla de transitividad, 153-154
regla de unión, 153
regla reflexiva, 153
regresión lineal, 426
regresión, 426, 496
rehacer
acciones tras un fallo, 341-343
fase, 341-343
paso, 351
sistemas de recuperación, 338-342
relación delta, 82
relación externa, 255
relación interna, 255
relación superclase-subclase, 133-134
relación temporal, 503-504
relación verdadera, 40, 41
relaciones vacías, 41-42
relevancia
buscadores Web, 440-441
clasificación por popularidad, 435-436
clasificación usando TF-IDF, 434-435, 438
concentradores, 437
enfoque TF-IDF, 434-438, 439-440
envenenamiento del motor de búsqueda, 437-438
PageRank, 436-438
prueba de adyacencia, 436-437
recuperación basada en la similitud, 435
uso de hiperenlaces, 190
realimentación de relevancia, 435
reloj lógico, 397
rendimiento
almacenamiento, 202, 213
almacenamiento en disco magnético, 198
aplicaciones web, 170-172
árboles B^+, 221-222
arquitectura del sistema, 362, 365, 376-377, 384
caché, 181
definición, 140
desarrollo de aplicaciones, 490, 494-495
desnormalización, 163
diseño de aplicaciones, 181-182
índices secuenciales, 221-222
media armónica, 495
mejora, 297-305
memoria principal, 528
Oracle, 552, 563
procesamiento paralelo, 181-182
registros del registro, 524
sistemas paralelos, 365
SQL Server de Microsoft, 600
tasa de transferencia, 198
tiempo de acceso, 196-200, 203, 205, 217-218, 241, 250-251, 383
tiempo de búsqueda, 196, 198-200, 205, 250, 257
tiempo de respuesta, 181 (véase también tiempo de respuesta)
tiempo de transacción, 164, 503
transacciones, 297, 305
dependencias de reunión, 161
repetir, 79
replicación
arquitectura del sistema, 369, 389-390
bases de datos distribuidas, 397
computación en la nube, 407-408
SQL Server de Microsoft, 598-599
replicación de instantánea, 598-599
replicación multimaestro, 397
replicación transaccional, 598-599
Representation State Transfer (REST), 178
requisitos de usuario, 7, 13
modelo E-R, 117, 134
rendimiento, 140-141
tiempo de respuesta, 140
resolución de desbordamiento, 259
respuesta a consultas, 441-442
restricciones
claves, 122-123
conservación de dependencias, 151-152
correspondencia de cardinalidad, 121-122, 124-125
DB2 de IBM, 571
definidas por condiciones, 135
definidas por el usuario, 135
descomposición, 159

de dominio, 5
disjuntas, 135
integridad, 2 (véase también restricciones de integridad)
modelo entidad-relación (E-R), 121-123, 135-136
parcial, 135
participación, 122
PostgreSQL, 538, 548-549
superposición, 135
total, 135
transacciones, 293, 525
UML, 139-140
restricciones de dominio, 5
restricciones de integridad diferidas, 60
restricciones de integridad inicialmente diferidas, 60
restricciones de integridad, 2, 27
añadir, 58
aserción, 61
asociación, 380
cláusula check, 59-61
clave externa, 59-60
clave primaria, 59
crear tabla, 58-59
dependencias funcionales, 58
diagramas de esquema, 23
diferidas, 60
ejemplos, 58
en una única relación, 58
no nulos, 58-60
referencial, 5, 23, 59-61, 67, 81, 293
tipos definidos por el usuario, 63
única, 59
violación durante una transacción, 60
XML, 474
restricciones de participación, 122
reunión (véase también reunión asociativa, reuniones)
álgebra relacional, 102-104, 107
análisis de coste, 257-258, 278-279
compleja, 261
condiciones, 51-52
de bucle anidado, 255-256 (véase también reunión de bucle anidado)
descomposición sin pérdidas, 155-156
dividida, 249-250, 379-381
equirreunión, 254-259, 261-262, 264,379, 384
estimación de tamaño, 276
estrategia de semirreunión, 402-403
externa, 52-54, 104-105, 261-262, 277
externa completa, 53-54, 104, 261-262
externa por la derecha, 53-54, 104, 261-262
externa por la izquierda, 52-54, 104, 261-262
filtrado, 565
fragmentar y replicar, 379-380
interna, 53-54, 279
mantenimiento de vistas, 283
mezcla, 256
mezcla híbrida, 258 DB2 de IBM, 576
minimización, 284
natural, 33-34, 39, 51 (véase también reunión natural)
Oracle, 556, 565
ordenación, 273-274
paralela, 378-381, 403
PostgreSQL, 548
predicción, 605
procesamiento de consultas, 254-262, 402-403
procesamiento distribuido, 402-403
relación externa, 255
relación interna, 255
reunión asociativa, 249-250, 258-260, 264-265, 279
reunión por mezcla, 256-257
theta, 271-272
tipos, 52-55
reunión asociativa híbrida, 260
reunión asociativa, 279
coste, 260
desbordamiento, 259
división recursiva, 249-250
encauzamiento doble, 264-265
híbridas, 260
principios, 258-259
procesamiento de consulta, 258-260
reunión de bucle anidado indexado, 255-256
reunión de bucle anidado, 508
DB2 de IBM, 576
optimización de consultas, 278-280
Oracle, 558
paralela, 379-381
PostgreSQL, 548-549
procesamiento de consultas, 255-256, 259, 261, 265
reunión externa, 52-55, 104-105, 261-262, 277
reunión externa completa, 53-55, 261-262
reunión externa por la derecha, 53-55, 104-105, 261-262
reunión externa por la izquierda, 53-55, 104-105, 261-262
reunión interna, 53-55, 279
reunión natural, 23-24, 39, 51
álgebra relacional, 102-104
condiciones, 51-52
consultas SQL, 33-34
externa completa, 53-55, 104, 261-262
externa por la derecha, 53-55, 104, 261-262
externa por la izquierda, 53-55, 104-105, 261
externa, 52-55
interna, 53-55, 279
sobre condiciones, 51-52
tipos, 52-55
reunión paralela, 378, 403
asociación, 380-381
bucle anidado, 381
división, 379-380
fragmentar y replicar, 379-380
reunión por mezcla híbrida, 258
reunión por mezcla, 256
reunión theta, 271-272
reuniones, 548
secuencialidad, 543-544
sentencias de operadores, 540
tipos, 537-539
transición de estados, 539-540
vacío, 543
visibilidad de tupla, 541
vuelta atrás, 543-544
revocación, 65, 67
robustez, 398
roles, 119-120
autorización, 65-66
diagramas entidad-relación, 125
Lenguaje unificado de modelado (UML), 139-140
RosettaNet, 499
Ruby on Rails, 174, 180
rutas de acceso, 251
rutinas de lenguaje externas, 79-80
salida forzada, 211, 336-337
Sarbanes-Oxley Act, 596
SAS (Serial attached SCSI), 197
SATA (ATA serie), 197-198
secuencialidad
aislamiento, 302-304
aislamiento de instantánea, 323-324
ajuste de rendimiento, 493
bases de datos distribuidas, 404-405
conflicto, 299-300
control de concurrencia, 309, 311, 313-314, 317-321, 323-324, 326-329
en el mundo real, 303
escritura ciega, 320
grafo de precedencia, 300
lectura de predicados, 326
Oracle, 562-563
orden de, 300-301
ordenación topológica, 300-301
PostgreSQL, 543-544
transacciones, 298-302, 303-304
vistas, 320
secuencialidad de vistas, 320
segunda forma normal, 162
seguridad, 3, 66
abstracción, 3-5
aislamiento, 293, 297-298, 301-304
ataque de hombre en medio, 183
ataque de persona en medio, 523

ataques de diccionario, 187
autenticación, 183-184
autorización, 5, 9, 184
base de datos privada virtual, 555
bloqueos, 309-319 (véase también bloqueos)
cifrado, 185-188, 555-556
claves, 22-23
contraseñas, 64, 72, 75-76, 169, 172, 174, 177, 183-184, 187, 210-211, 410
control de concurrencia, 310-331 (véase también control de concurrencia)
diseño de aplicaciones, 181-188
escrituras externas observables, 296-297
gestor de integridad, 9
guion de sitio cruzado, 182-183
identificación única, 185
independencia de los datos físicos, 3
información de usuario final, 184
inyección de SQL, 181-182
método GET, 183
Oracle, 554-555
privacidad, 181, 185, 188, 390, 409, 523
pruebas de auditoría, 185
sistemas de copia de seguridad remota, 352-353
sistemas de una sola firma, 183
SQL Server de Microsoft, 596-597
transacciones de larga duración, 525-528
Lenguaje de marcas de aserción de seguridad (SAML), 183

selección
álgebra relacional, 97-98
búsqueda lineal, 251
comparaciones, 252-253
compleja, 252-253
conjunción, 252-253
disyunción, 252-253
equivalencia, 271-273
exploración de archivos, 251-252, 255, 264
identificadores, 253
índices, 251-252
mantenimiento de vistas, 283
paralelismo intraoperación, 381

select, 163 (véase también select seguido de cláusulas específicas)
clasificación, 86
consultas de SQL básicas, 29-34
especificación de atributos, 35
funciones agregadas, 38-41
miembros del conjunto, 40-41
operación rename, 34-35
operaciones de cadena, 35-36
operaciones de conjuntos, 36-37
privilegios, 64-66
reunión natural, 33-34
sobre relaciones múltiples, 30-33
sobre una relación única, 29-30
valores *null*, 37-38

select all, 30

select distinct, 29-30, 38, 41, 57

select-from-where
actualización, 44-45
borrado, 33-34
función/procedimiento de escritura, 78-80
herencia, 451-453
inserción, 44-45
manejadores de tipos, 451-456
reunión natural, 31-34, 39, 51-54
subconsultas anidadas, 40-43
subexpresión de reunión, 31-34, 39, 51-55
transacciones, 303-304
vistas, 55-58

sensibilidad, 426

sentencia from
consultas básicas de SQL, 29-34
en relaciones múltiples, 30-32
funciones de agregación, 38-40
operación de renombrado, 34-35
operaciones de cadena, 35-36
operaciones de conjunto, 36-37
reunión natural, 33-34
sobre una sola relación, 29-30
subconsultas, 42-43
valores nulos, 37-38

sentencia open, 76-77

sentencias ejecutables, 74

sentencias preparadas, 72-74

Sequel, 27

servicios Web, 178, 571-572

servidores de nombres, 391

servidores de vídeo, 511

servidores Web, 171-172

servlets
ciclo de vida, 174
ejemplo de, 173
guiones del lado del cliente, 175-176
guiones del lado del servidor, 174-175
sesiones, 173-174
soporte, 174

Sherpa/PNUTS, 408

Simple API for XML (SAX), 476

sin agrupamiento, 218

sinónimos, 438-439

sistema(s) (véase también sistemas específicos)

sistema de intercambio, 523

sistema de malla, 366

Sistema de posicionamiento global (GPS), 509

sistema en bus, 366

sistema servidor de transacciones, 363-364

sistemas de bases de datos en memoria principal, 336

sistemas de colas, 489-490

sistemas de copia de seguridad remota, 334, 352-353, 400, 519-520

sistemas de directorio, 409-413

sistemas de extracción de información, 441-442

Sistemas de planificación de recursos empresariales (ERP), 521

sistemas de recuperación, 82, 295, 354-355, 513
acceso a datos, 336-337
acciones tras un fallo, 341-342
aislamiento de instantánea, 338-339
algoritmo, 341-342
almacenamiento, 335-337, 340-341, 345-346
ARIES, 348-352
atomicidad, 337-341
bases de datos distribuidas, 393-394
control de concurrencia, 338-339
copia de seguridad remota, 336, 352-353, 400, 519
DB2 de IBM, 572-573, 579-580
deshacer, 338-343
divisiones, 556-558
espejo de la base de datos, 595-596
esquema de copia en la sombra, 338
fallos, 335-336, 345-346
fallos de disco, 335
flujos de trabajo, 521
gestión de memoria intermedia, 343-345
liberación temprana de bloqueos, 345-348
modificación de la base de datos, 337-338
números de secuencia de registro (LSN), 349
operación de deshacer lógico, 345-348
Oracle, 561-563
política de apropiación/no apropiación, 343
política de forzar/no forzar, 343-344
PostgreSQL, 544-545
puntos de comprobación, 340-341, 344-345
registros de registro, 337-338, 339-341, 343
Regla de registro de escritura adelantada (WAL), 343-344, 544-545
rehacer, 338-343
SQL Server de Microsoft, 593-595
suposición de fallo-parada, 335
terminación con éxito, 336
transacciones de larga duración, 526
vuelta atrás, 338-341

sistemas de soporte a la decisión, 419-420

sistemas de transacciones en tiempo real, 525-526

sistemas distribuidos
aumento de la sobrecarga de procesamiento, 370
autonomía, 369

compartición de datos, 369
coste de desarrollo de software, 370
disponibilidad, 369
ejemplos, 369
estado de listo, 370
implementación, 369-370
mayor potencial de errores, 370
nodos, 368
Protocolo de compromiso de dos fases (2PC), 369-370
replicación, 369
sitios, 368
transacciones globales, 368
transacciones locales, 368
sistemas heredados, 497
sistemas multibase de datos, 403-405
sistemas paralelos
coste de inicio, 366
disco compartido, 366-367
escalado, 365-366
grano fino, 365
grano grueso, 365
interferencia, 366
jerárquicos, 366-367
masivamente paralelos, 365
memoria compartida, 366-368
redes de interconexión, 366-367
rendimiento, 365
sin compartición, 366-367
sistemas servidor
basados en la nube, 365
categorización, 362-363
cliente-servidor, 362
servidor de transacciones, 363-364
servidores de datos, 363-365, 368
sistemas servidores de datos, 363-365, 368
sitio principal, 352
sitio secundario, 352
sobreajuste, 425
sobrecarga, 458-459
sobrecarga en el espacio, 217-218, 222, 241
software RAID, 204
Solaris, 569
soporte a la decisión, 495
SQL, entorno, 64
SQL/DS, 14
SQL/XML standard, 478-479
SQL (Structured Query Language), 5-7, 27, 67, 93, 271
acceso desde lenguajes de programación, 71-73
actualización, 44-45
álgebra relacional, 98, 107
autorización, 27, 64-67
autorización del nivel de aplicación, 183-185
avanzado, 71-93
bases de datos basadas en objetos, 449-462
blob, 62, 74, 208, 231, 478, 571-572, 602
carga en masa, 488-489
catálogos, 64
cláusula select, 35
clob, 62, 74, 208, 231, 477-478, 570-571
como lenguaje estándar de bases de datos relacionales, 27
creación de índices, 61-62, 243-244
crear una tabla, 28-29, 63-64
DB2 de IBM, 570-572, 576
delete, 43-44
desarrollo rápido de aplicaciones (RAD), 179
descripción general, 27
dinámico, 27, 71, 76-77
disparadores, 80-83
embebido, 27, 71, 76-77, 363
entidad, 178
entorno, 20, 64
escritura de funciones, 77-78
escritura de procedimientos, 77-80
especificaciones de tiempo, 504
esquemas, 23, 27-29, 63-64, 66
expresiones de reunión, 33, 55 (véase también reunión)
falta de autorización de grano fino, 184-185
funciones de agregación, 38-40, 85-87
generación de informes, 180-181
herencia, 451-453
inserción, 44-45
intermedio, 51-57
inyección, 181-182
JDBC, 71-74
Lenguaje de definición de datos (DDL), 27-29, 46
Lenguaje de manipulación de datos (DML), 27, 46
lenguajes de programación persistente, 457-461
Management of External Data (MED), 510
minería de datos, 12
modificación de la base de datos, 43-45
MySQL, 14, 35, 48, 72, 535, 549
niveles de aislamiento, 302-304
normas, 498
objeto ResultSet, 71-72, 74-75, 177, 179-180, 224
objetos de tipo grandes, 62
ODBC, 74-76
OLAP, 88-92
operación rename, 34-36
operaciones de cadenas, 35
operaciones de conjuntos, 36-37
PostgreSQL, 14 (véase también PostgreSQL)
predicados de la cláusula where, 76
procesamiento de consultas, 249 (véase también procesamiento de consultas)
programas de aplicación, 6-7
restricciones de integridad, 27, 58-61
revocación de privilegios, 67
roles, 65-66
seguridad, 181-183
sentencias preparadas, 72-74
sintaxis no estándar, 80
sistemas de soporte a la decisión, 419-420
SQL Server de Microsoft, 585-605
subconsultas anidadas, 40-43
System R, 14, 27
tipos básicos, 28
tipos date/time, 61-62
tipos de array, 453-455
tipos definidos por el usuario, 62-63
tipos multiconjunto, 453-455
transacciones, 27, 57-58, 363 (véase también transacciones)
transferencia de privilegios, 66-67
tuplas, 35-36 (véase también tuplas)
valores nulos, 37-38
valores predeterminados, 61
variaciones de Oracle, 552-553
vistas, 27, 55-58, 65-66
volcado, 345-346
SQL Access Group, 498
SQL dinámico, 27, 71, 76-77
SQL embebido, 27, 71, 76-77
SQL Plan Management, 560-561
SQL Profiler, 586-587
SQL Security Invoker, 66
SQL Server Analysis Services (SSAS), 604-605
SQL Server Broker, 603-604
SQL Server de Microsoft, 493, 530, 585, 605
acceso a datos, 596-597
actualizaciones, 588-589, 592
administración de memoria, 595-596
aislamiento de instantánea, 593-594
ajuste, 585-587
almacenamiento, 589-591
arquitectura del sistema, 595-597
bancos de hebras, 595
bloqueos, 593-594
compilación, 591
compresión, 591
control de concurrencia, 593-596
desarrollo, 585
disparadores, 588-589
espejo de base de datos, 595-596
grupos de archivos, 589-590
herramientas de administración, 585-587
herramientas de diseño, 585-587
indexado, 588-590
inteligencia de negocio, 603-605
lectura adelantada, 590-591

minería de datos, 605
particiones, 590
procesamiento de consultas, 585-589, 591-593, 597-598
programación de servidor en .NET, 599-601
Query Editor, 585-586
recuperación, 593-595
reordenación, 591-592
replicación, 598-599
rutinas, 588
seguridad, 596-597
soporte de XML, 601-603
SQL Profiler, 586-587
SQL Server Broker, 602-604
SQL Server Management Studio, 585-587
tablas, 590
tipos, 600-601
tipos de datos, 587-588
unidades de páginas, 589-590
variaciones de SQL, 587-589
Windows Mobile, 585
SQL Server Integration Services (SSIS), 604-605
SQL Server Management Studio, 585-587
SQL Server Reporting Services (SSRS), 604-605
SQL Transparent Data Encryption, 596
SQL Tuning Advisor, 559-560
SQLJ, 77
SQLLoader, 488, 566
sqlstate, 79
Starburst, 569
Storage area network (SAN), 197-198, 370
subconsultas anidadas
cláusula from, 42-43
cláusula with, 43
desarrollo de aplicaciones, 488, 495
escalar, 43-44
operaciones de conjuntos, 40-42
optimización, 280-282
relaciones vacías, 41-42
tuplas duplicadas, 42
subconsultas correlacionadas, 41
sufijo, 412
sum, 38, 56, 91, 104-105, 262-263, 539
sumas de comprobación, 197
superclave, 21-22, 122-123, 150-151
superusuario, 64
Swing, 180
Sybase, 585
sysaux, 558
System R, 14, 27, 569
tabla de páginas sucias, 349-352, 594-595
tabla(s), 6
.NET Common Language Runtime (CLR), 600-601
DB2 de IBM, 572-573
divisiones, 556-558
filtrado, 565
materializadas, 577-578
modelo relacional, 19-21, 23-24
Oracle, 553-555, 565-566
SQL Server Broker, 603
SQL Server de Microsoft, 588, 590
Tablas de consulta materializadas (MQT), 577-578, 581
tablas de hechos, 420-421
tablas de transición, 81-82
Tablas organizadas por índices (IOT), 554-555
Tapestry, 180
tasa de transferencia de datos, 197-198
Tcl, 80, 535-536, 540
técnica de modificación inmediata, 338
Teradata Purpose-Built Platform Family, 378-379
Tercera forma normal (3NF)
algoritmos de descomposición, 158-159
bases de datos relacionales, 152-153, 158-159
Término de frecuencia (TF), 434
tiempo con zona de tiempo, 504
Tiempo coordinado universal (UTC), 504
tiempo de búsqueda medio, 197-198, 205, 250, 257
tiempo de compromiso, 353
tiempo de recuperación, 353
tiempo de respuesta
almacenamiento, 202, 524-526
arquitectura del sistema, 365, 375-377
control de concurrencia, 320
diseño de aplicaciones, 181, 490, 495
modelo E-R, 140
Oracle, 559-560
planes de evaluación de consultas, 251
procesamiento de consultas, 251
SQL Server de Microsoft, 603
transacciones, 297
restricción, 67, 156
tiempo de terminación, 494
tiempo de transacción, 164, 503
tiempo medio entre fallos (MTTF), 198
tiempo medio de respuesta, 297
tiempo válido, 164
tiempos de búsqueda, 196-197, 200, 205, 250, 257
tipo más específico, 452
tipos, 480, 552
.NET Common Language Runtime (CLR), 600-601
abstractos de datos, 537
ajuste de rendimiento, 493
área extensa, 370-372
array, 453-455
base, 537
bases de datos basadas en objetos, 451-456
blob, 62, 74, 208, 231, 476, 571-572, 602
clob, 62, 74, 208, 231, 476-478, 570-572
compuestos, 537
datos complejos, 449, 451 (véase también tipos de datos complejos)
DB2 de IBM, 570-571
definición de tipo de documento (DTD), 469-470
definición de usuario, 62-63
enumerados, 537
herencia, 451-453
identidad de objetos, 455-456
más específicos, 452
multiconjunto, 453-455
no estándar, 537-538
Oracle, 551-552
polimórficos, 537
PostgreSQL, 537-539
pseudotipos, 537
referencia, 455-457
referencia única, 460
SQL Server de Microsoft, 537-538, 600-601
XML, 469-472, 475
tipos abstractos de datos, 537
tipos de datos complejos, 13-14, 62, 503
arquitectura del sistema, 406
bases de datos basadas en objetos, 449-450, 456, 459-462
formas normales, 450
modelo entidad-relación (E-R), 449-450
palabras clave, 450-451
tipos de objetos grandes, 62
tipos definidos por el usuario, 62-63
tipos distintos, 38, 62-63
tipos estructurados, 62-63, 451-452
tipos multiconjunto, 453-455
tipos polimórficos, 537
Tomcat, 174
trabajo de compromiso, 57
trampa de araña, 440
transacciones, 15, 289, 305, 529
abortadas, 295-296, 301
acciones tras un fallo, 341-342
activas, 295
aislamiento, 293, 297-298, 301-304 (véase también aislamiento)
ajuste de rendimiento, 493-494
ataque de hombre en medio, 524
atomicidad, 10-11, 293-296, 301-302 (véase también atomicidad)
bases de datos basadas en objetos, 449-462
bases de datos distribuidas, 391-392
bases de datos en memoria principal, 524-525
bases de datos paralelas, 375-385

bloqueos, 309-312, 313-320 (véase también bloqueos)
comercio electrónico, 522-524
comienzo/fin de operaciones, 293
como unidad de programa, 293
compensación, 295, 527-528
comprobación de restricciones, 293
comprometidas, 57, 295-296, 298, 301, 322-323, 339, 353, 392-395, 524, 580
computación en la nube, 407-409
concepto, 293-294
consistencia, 10, 293-295, 297-298, 302-303, 305 (véase también consistencia)
control de concurrencia multiversión (MVCC), 541-545
control de concurrencia, 309, 330, 593-595 (véase también control de concurrencia)
definición, 10, 293
disponibilidad, 398-401
durabilidad, 10-11, 293, 295-296 (véase también durabilidad)
escritura externa observable, 296-297
esquema de copia en la sombra, 338
esquema multiversión, 321-323
estados, 296
estructura de almacenamiento, 295-296
fallo de, 295, 335
fallos, 293
flujo de trabajo, 394-395, 519-522
gestión de recuperación, 10-11
globales, 368, 391, 404-405
grafo de espera, 315-316
inanición, 311
larga duración, 525-528 (véase también transacciones de larga duración)
lenguajes de programación persistentes, 459
locales, 368, 391, 404-405
marcas de tiempo, 318-319
matadas, 296
mensajería persistente, 394
minería de datos, 422-429
modelo simple, 294-295
multinivel, 527
multitarea, 517-519
no comprometidas, 302
operaciones de lectura/escritura, 304-305
planificaciones recuperables, 301
planificación sin cascada, 301-302
política de apropiación/no apropiación, 343
políticas de forzar/no forzar, 343-344
PostgreSQL, 541-545
procesamiento avanzado, 527-529
protocolo de compromiso de dos fases (2PC), 369-370
protocolos de compromiso, 392-395
registros del registro, 337--341
Regla del registro de escritura adelantada (WAL), 343-344
reglas de asociación, 427-428
reinicio, 296
secuencialidad, 298-304
sentencias de SQL, 304-305
sistemas de copia de seguridad remota, 352-353
sistemas de recuperación, 295-296 (véase también sistemas de recuperación)
sistemas de tiempo real, 525
SQL Server Broker, 603-604
validación, 320-321
violación de restricciones de integridad, 60
vuelta atrás, 57, 341, 346-348, 351-352
transacciones de larga duración
anidamiento, 526-527
bloqueo de dos fases, 526
control de concurrencia, 526-527
datos no comprometidos, 525
elementos de implementación, 527-528
ejecución no secuenciable, 526
marcas de tiempo, 526
multinivel, 526-527
protocolos basados en grafos, 526
recuperabilidad, 526
registro de operación, 528
rendimiento, 526
subtareas, 525
transacciones de compensación, 527-528
Transacciones por segundo (TPS), 495-496
Transaction Processing Performance Council (TPC), 495-496
TransactSQL, 77
transferencia de control, 352
transferencia de prestigio, 435-436
transformaciones
álgebra relacional, 270-274
ejemplos, 272-273
optimización de consultas, 270-274
ordenación de reunión, 273-274
reglas de equivalencia, 271-272
XML, 472-475
transiciones de estado, 539
transparencia, 390-391, 402
tuplas, 19-20
actualización, 44-45
álgebra relacional, 97-107, 271-274
bases de datos paralelas, 375-385
borrado, 43-44
cálculo relacional de dominios, 109-110
clasificación, 85-87
duplicadas, 42
encauzamiento, 263-265
estructuras de consulta, 31
funciones de agregación, 38-40
generación perezosa, 263-264
inserción, 44-45
mostrar ordenadas, 35-36
operaciones de conjuntos, 36-37
optimización de consultas, 269-286
PostgreSQL, 541-545
procesamiento de consultas, 249-266
producto cartesiano, 24
reunión, 255-256 (véase también reunión)
ventanas, 87
vistas, 55-59
Ultra320 SCSI, interfaz, 198
Ultrium, formato, 205
único, 42
descomposición, 159-160
restricciones de integridad, 59
unión, 36-37, 82, 98-99
Universal Description, Discovery, and Integration (UDDI), 480
Universal Serial Bus (USB), 195
Universidad de California, Berkeley, 14, 535
universidades, 1
almacenamiento, 206, 208-209
arquitectura del sistema, 369, 390, 410
bases de datos para, 2-9, 13-14
control de concurrencia, 325
diseño de aplicaciones, 169, 177, 184
diseño de base de datos relacional, 147-151, 160, 163-164
diseño de bases de datos, 7-8
indexado, 218, 234, 243
modelo E-R, 118-124, 126-127, 132-134
modelo relacional, 20-23
optimización de consultas, 272, 274, 281
procesamiento de consultas, 262
sistema de recuperación, 336
SQL, 28-29, 31-33, 35, 44, 56-60, 65-68, 76-77, 83, 85-86, 88, 101-102
transacciones, 304
Unix, 35, 199, 332, 338, 535, 549, 569-570, 577, 585
using, 51
utilización, 297
vacío, 543
validación, 327-328
aislamiento de instantánea, 322-323
clasificadores, 426-427
control de concurrencia, 319-321
fases, 320
primero en actualizar gana, 323
primero en comprometer gana, 322-323
secuencialidad de vistas, 320
sistemas de recuperación, 338-339
transacciones de larga duración, 526
valor(es) de estado, 539
valor falso, 40, 92
valores *null*, 8, 37-38
agregación, 40
atributos, 121-122

datos temporales, 163-164
decodificar, 92
descorrelación, 559
OLAP, 90
organización de archivos, 206-213
restricciones de integridad, 58-61
reunión externa por la derecha, 104-105
reunión externa por la izquierda, 104
simplificación de consultas, 591-592
tipos definidos por el usuario, 63
valores predeterminados, 60, 62-63, 65, 191, 425, 452, 469, 471, 537
varchar, 28-29
variables de transición, 81
VBScript, 174
vector de división, 375-376
vector de división por rangos, 376
velocidad sublineal, 365-366
ventanas, 87
verdaderos negativos, 426
visibilidad de tuplas, 541
vistas, 55
actualización, 56-58
ajuste de rendimiento, 491-492
autorización, 66
borrado, 57
con opción de comprobación, 57
consultas de SQL, 55-56
creación de vistas, 55-57
cubo, 581-582
definición, 55-56
inserción en, 56-57
mantenimiento, 56
mantenimiento diferido, 491-492
mantenimiento inmediato, 491-492
materializadas, 56 (véase también vistas materializadas)
mezcla compleja, 558-559
vistas materializadas, 56, 282
agregación, 283-284
ajuste de rendimiento, 491-492
DB2 de IBM, 577-578
mantenimiento, 282-284
operación de reunión, 283
optimización de consulta, 284
Oracle, 557-559, 565
proyección, 283
replicación, 598-599
selección, 283
selección de índice, 284
Visual Basic, 76, 80, 179-180, 587
VisualWeb, 179
volcado, 345-346
volúmenes colección, 454
vuelta atrás, 77
ARIES, 351
control de concurrencia, 311-312, 314-316, 319, 321-322, 330
DB2 de IBM, 580
deshacer, 338-341
en cascada, 311
marcas de tiempo, 319-320
operaciones lógicas, 346-348
PostgreSQL, 543-544
sistemas de copia de seguridad remota, 353-354
sistemas de recuperación, 338-341
transacciones, 341
Web semántica, 439
Weblogic, 174
WebObjects, 180
WebSphere, 174
Wi-Max, 512
Windows Mobile, 585
WordNet, 439
World Wide Web, 14, 415
aplicaciones Web, 179-180
arquitectura en tres capas, 144
cifrado, 185-188
cookies, 172-174, 182-183
diseño de aplicaciones, 170-172
Lenguaje de marcado de hipertexto (HTML), 171
Localizador uniforme de recursos (URL), 170-171
Protocolo de transferencia de hipertexto (HTTP), 170-171, 173, 178, 182-183, 188
recuperación de información, 433 (véase también recuperación de información)
seguridad, 181-188
servicios de procesamiento, 178
servidores Web, 171-173
Simple Object Access Protocol (SOAP), 480
XML, 480
World Wide Web Consortium (W3C), 438, 499
X.500, protocolo de acceso a directorio, 410
XML (Lenguaje de marcado extensible), 14, 76, 174, 481
actualizaciones, 602
almacenamiento, 476-479
anidamiento, 12, 446, 465-469, 470-473, 474-475, 476
aplicaciones, 479-481
bases de datos relacionales, 476-478
como formato dominante, 466
concepto de marca, 465-466
consultas, 472-476
contexto textual, 467
DB2 de IBM, 570
definición de tipo de documento (DTD), 469-470
encapsulamiento, 499-500
esquema de documento, 469-472
estructuras de datos, 467-469
etiquetas, 465-468, 470, 472, 474, 480
flexibilidad de formato, 466
formatos de intercambio de datos, 479-480
HTML, 465
Interfaz de programación de aplicaciones (API), 475-476
mapas relacionales, 477
mediación de datos, 480-481
modelo en árbol, 472
normas para, 499-500
normas SQL/XML, 478-479
ordenación, 475
procesamiento de archivos, 465
reunión, 474
servicios web, 479-480
Simple Object Access Protocol (SOAP), 479-480
SQL Server de Microsoft, 601-603
transformaciones, 472-476
valores clob, 476-477
XML DB de Oracle, 552
XML Schema, 470-472
xmlagg, 478
xmlattributes, 478
xmlconcat, 478
xmlelement, 478
xmlforest, 478
XMLIndex, 552
XMLType, 552
XPath, 552
almacenamiento, 476-478
consultas, 472-473
esquema de documento, 471
XQuery, 14, 472
almacenamiento, 476-478
consultas anidadas, 474-475
expresiones FLWOR, 473-474
funciones, 475
Oracle, 552
ordenación de resultados, 475
reuniones, 45-46
SQL Server de Microsoft, 602-603
tipos, 475
transformaciones, 473-476
XSLT, 552
Yahoo, 176, 405
zona de tiempo, 61-62, 504